Universitat Politècnica de Catalunya, UPC,

Doctoral Thesis

Data Center's Telemetry Reduction and Prediction through Modeling Techniques

Author:
Shuja-ur-Rehman Baig

Supervisors:
David Carrera and Josep Lluís Berral

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Department of Computer Architecture

September 3, 2019

Abstract

Nowadays, Cloud Computing is widely used to host and deliver services over the Internet. The architecture of clouds is complex due to its heterogeneous nature of hardware and is hosted in large scale data centers. To effectively and efficiently manage such complex infrastructure, constant monitoring is needed. This monitoring generates large amounts of telemetry data streams (e.g. hardware utilization metrics) which are used for multiple purposes including problem detection, resource management, workload characterization, resource utilization prediction, capacity planning, and job scheduling. These telemetry streams require costly bandwidth utilization and storage space particularly at medium-long term for large data centers. Moreover, accurate future estimation of these telemetry streams is a challenging task due to multi-tenant co-hosted applications and dynamic workloads. The inaccurate estimation leads to either under or over-provisioning of data center resources. In this Ph.D. thesis, we propose to improve the prediction accuracy and reduce the bandwidth utilization and storage space requirement with the help of modeling and prediction methods from machine learning. Most of the existing methods are based on a single model which often does not appropriately estimate different workload scenarios. Moreover, these prediction methods use a fixed size of observation windows which cannot produce accurate results because these are not adaptively adjusted to capture the local trends in the recent data. Therefore, the estimation method trains on fixed sliding windows use an irrelevant large number of observations which yields inaccurate estimations. In summary, we C1) efficiently reduce bandwidth and storage for telemetry data through real-time modeling using Markov chain model. C2) propose a novel method to adaptively and automatically identify the most appropriate model to accurately estimate data center resources utilization. C3) propose a deep learning-based adaptive window size selection method which dynamically limits the sliding window size to capture the local trend in the latest resource utilization for building estimation model.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisors David Carrera Perez and Josep Lluis Berral for the continuous support of my Ph.D. study and research. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors and mentors for my Ph.D. study.

Besides my advisors, my sincere thanks also go to Dr. Waheed for his support and Dr. Abdul Karim Erradi for offering me the summer internship opportunities in his group and leading me working on diverse exciting projects. I thank my fellow lab mates in Data centric Group: Jorda Polo, Nicola Cadenelli, Cesare Cugnasco, Marcelo Amaral, David Buchaca, Alberto Gutierrez, Aaron Call. And last but not the least, I would like to thank my family: my parents, brothers and sisters, wife, and children.

Contents

List of Figures

List of Tables

List of Abbreviations

PR	**P**olynomial **R**egression
MM	**M**arkov chain **M**odel
K-S	**K**olmogorov **S**mirnov
DTW	**D**ynamic **T**ime **W**arping
LR	**L**inear **R**egression
SVR	**S**upport **V**ector **R**egression
RR	**R**idge **R**egression
LASSO	**L**east **A**bsolute **S**hrinkage and **S**election **O**erator **R**egression
EN	**E**lastic **N**et
NNLS	**N**on **N**egative **L**east **S**quare
K-NN	K **N**earest **N**eighbors
MLP	**M**ulti **L**ayer **P**erceptron
RDF	**R**andon **D**ecision **F**orest
T-R	**T**elemetry **R**eductor
NR	**N**ot **R**educed
NA	**N**ot **A**ccepted
AMS	**A**daptive **M**odel **S**elector
GBM	**G**radient **B**oosting **M**achine
RMSE	**R**oot **M**ean **S**quare **E**rror
MAE	**M**ean **A**bsolute **E**rror
GBT	**G**radient **B**oosting **T**ree
TPR	**T**rue **P**ositive **R**ate
FPR	**F**alse **P**ositive **R**ate
TNR	**T**rue **N**egative **R**ate
FNR	**F**alse **N**egative **R**ate
ROC	**R**eceiver **O**perator **C**haracteristics

This thesis is dedicated to my parents, loving wife and children for their endless support, love and encouragement.

Chapter 1

Introduction

1.1 Introduction

Technological advances enable users to access computing resources over the Internet quickly and cost-effectively for developing new solutions with intensive data and computing requirements. Most of the increasing demands are driven by novel applications on social networks, Big Data analytics, Smart Cities, and the Internet of Things (IoT), also e-commerce and Business-to-Business processes. Cloud Computing is one of the central technologies enabling end-users to obtain such resources by following *pay-as-you-go* models. To ensure acceptable levels of Quality of Service and user experience, cloud providers distribute data center resources across the geography to enhance proximity to the user and clients, also by virtualizing resources allowing service customization. The most common model used is precisely Infrastructure-as-a-Service (IaaS). IaaS provides users with excellent control of leased resources, allowing them to optimize their usage accordingly to their needs. Also, thanks to virtualization and holistic data center management, cloud providers can manage resources to maximize the data center utilization while minimizing infrastructure usage [105].

These large-scale data centers consist of thousands of servers, organized in racks and interconnected to offer services to a broad set of users. Such data centers generate large and continuous streams of telemetry data that are logged and analyzed for multiple purposes including resource management, workload characterization, resource utilization prediction, capacity planning and real-time analytics [104, 10, 63, 133]. Typical telemetry streams contain time-series data about hardware utilization metrics including CPU, memory, I/O, bandwidth, context switches, interrupts, cache misses, and cycles per instruction. These telemetry metrics are used in various applications. For example, CPU, memory, network bandwidth, instructions per cycle (IPC), cache miss rates, branch predictor statistics, and power consumption utilization data is used to forecast energy consumption and reduction [56, 34]. CPU, memory, disk, and network consumption are used for data center management and resource prediction [127, 117, 35, 100].

The first challenge is that this generation of continuous telemetry streams from all computing and storage nodes poses a significant challenge within the data center in terms of bandwidth consumption and storage requirements. As an example, considering telemetry collection on a data center consisting of $10,000$ computing nodes and collecting 12 different measured metrics every second while dedicating 4 bytes per metric, would require nearly $40GB$ storage per day, this is more than $1TB$ of storage per month, plus the meta-data overheads for time-series traceability. Traditionally, data compression solutions to reduce the data increase the time to collect

telemetry data from the computing nodes. Increased interval time does not allow capturing fine-grained resource consumption and may not precisely reflect the resources usage. Also, using compression techniques on floating-point values cannot reduce the size significantly and preserve the full precision [28, 14, 53]. Unfortunately, telemetry streams in many data centers show low or no smoothness and high variation in data. In such cases, even state-of-the-art floating-point compression algorithms are still not sufficient to compress the data [102]. Moreover, on large-scale data centers with millions of hosted applications, the usefulness of telemetry and profiling comes from knowing the behavior patterns more than the exact metrics. The precision is reduced when data is aggregated in bigger time periods. Hence, telemetry time-series data reduction methods need to preserve statistical properties capturing hardware utilization behaviors specifically burstiness,the rapid growth of utilization, and abnormal hardware utilization patterns.

The second challenge when managing data centers is that in most situations multitenancy environments, co-hosted applications, and/or dynamic workloads, estimating resources *a-priori* becomes hard or inaccurate. Accurate data center's resource utilization forecasting is important for various reasons including resource management [97, 43, 99, 68, 46], energy saving , cost prediction and consolidation of virtual machines (VMs) [33, 2, 83], and capacity planning [17, 112]. An accurate estimation of resource demands can greatly help to optimize the operational cost of the applications for the end uses and also helpful for the data centers to reduce the cost to offer the facility for a large number of users. For example, the data center users who can dynamically manage the resources to minimize the operating cost while maintaining the good quality of service [52], whereas the providers can increase the profit by maximizing the use of available resources [123]. There have been several efforts to build estimation methods for cloud resource utilization. For example, some recent works [58, 97] use ensemble-based methods to predict the resource utilization of data centers. The ensemble-based approach uses multiple time-series prediction and machine learning method together to produce an output, and some of the existing estimation methods are based on a single model which often does not appropriately estimate different workload scenarios. Moreover, these prediction methods use a fixed size of observation windows which cannot produce accurate results because these are not adaptively adjusted to capture the local trends in the recent data. Therefore, the estimation method trains on fixed sliding windows use an irrelevant large number of observations which yields inaccurate estimations.

To address these challenges, we formulate the thesis statement of this Ph.D. proposal as follow.

It is possible to improve the storage, bandwidth utilization, and prediction accuracy of data center telemetry by applying modeling techniques.

We selected modeling as a principal methodology because models can represent data without actually containing it. These models help us in reducing the bandwidth utilization and storage space requirements significantly, which is the first part of the thesis statement. Additionally, modeling techniques allow to reconstruct and forecast such data, so we apply these techniques to improve the prediction accuracy of data center telemetry saving, up to a certain point, on monitoring efforts. For this purpose, we explore different types of machine learning techniques, also adaptive methods for selection of those models, and adaptive identification of time window sizes for these prediction models, being this the second part of the thesis. So in summary, to achieve

thesis goal, we divide the problem into three incremental research steps as follow: i) Design and evaluate the telemetry reduction system to manage the bandwidth and storage space requirement better. ii :) Design and evaluate the resource prediction system based on adaptively selecting the prediction model iii :) Design and evaluate the resource prediction system based on adaptively selecting the window size for prediction models.

1.2 Contributions

The focus of this thesis is to improve telemetry prediction accuracy and reduce the storage space and bandwidth utilization requirements in the data centers environment. To achieve it, we divide the goal into following three contributions and Figure 1.1 also summarize the main contribution of the thesis.

C1: Design and development of telemetry data reduction and reconstruction method using Markov Chain Models

C2: Design and development of adaptive prediction model selector method which dynamically selects the best prediction model for estimating and forecasting cloud resource utilization.

C3: Design and development of an adaptive window size selector method for prediction models which dynamically identify the best sliding window size to train the prediction model for estimating and forecasting cloud resource utilization

Next, we further detail the three contributions of this thesis.

1.2.1 C1: Telemetry Reduction and Reconstruction using Markov Chain Models:

The **first contribution** of this thesis addresses the issue of telemetry data, which is generated by large-scale data centers. These data centers typically consist of thousands of servers organized in interconnected racks and generate a large amount of telemetry data continuously. Typically these telemetry streams consist of time series data which include CPU, memory, I.O, and other hardware utilization metrics. These metrics are used for multiple purposes, including capacity planning, resource management, workload characterization, resource utilization prediction, real-time analytics, and many more. This generation of continuous telemetry streams requires costly bandwidth utilization and storage space for these kinds of data centers. The first contribution addresses this problem by proposing and evaluating a system to efficiently reduce bandwidth and storage for telemetry data through real-time modeling using Markov chain-based methods. Our proposed solution was evaluated using real telemetry data sets and compared with Polynomial regression methods for reducing and reconstructing data. Experimental results show that data can be lossy compressed up to 75% for bandwidth utilization and 95.33% for storage space, with reconstruction accuracy close to 92%.

The work performed in this area has resulted in the following publication:

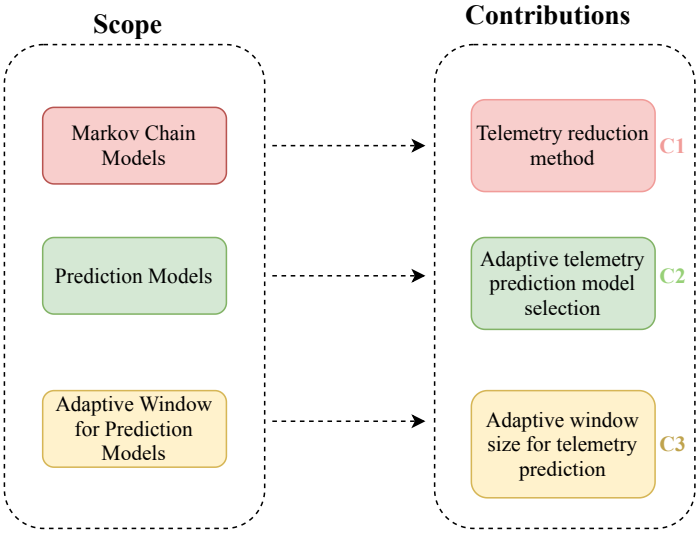


Figure 1.1: Summary of Contributions. The first contribution (C1) is the design and development of telemetry reduction method for data centers by using Markov Models. The second contribution (C2) is the design and development of adaptive prediction model selector method for data centers telemetry prediction by using machine learning prediction models. The third contribution (C3) is the design and development of an adaptive window size selector method for prediction models by using deep learning.

- Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, David Carrera, "Real-time Data Center's Telemetry Reduction and Reconstruction Using Markov Chain Models," IEEE Systems Journal

In this contribution, we focused on modeling techniques to reduce the telemetry data for data centers using Markov Chain Models, towards the next contribution, where we apply learning for modeling and prediction of data center's telemetry stream, focusing on improving the prediction accuracy in an adaptive way.

1.2.2 C2: Adaptive Prediction Model Selector:

The **second contribution** of this thesis addresses the issue of telemetry prediction. Due to heterogeneous architecture, multi-tenant co-hosted applications, and dynamic workloads, the accurate estimation of data center resource utilization is a challenging task. Accurate estimation of future resources utilization helps in better capacity planning, job scheduling, workload placement, proactive auto-scaling, and load balancing. The inaccurate estimation leads to either under or over-provisioning of data center resources. In this contribution, we address this problem by proposing a novel method to adaptively and automatically identify the most appropriate model to estimate data center resources utilization accurately. The proposed approach trains a classifier based on statistical features of historical resource utilization observations to decide the appropriate prediction model to use for given resource utilization observations for a specific time interval. We evaluated our approach with multiple baseline methods and real data sets. The experimental evaluation shows that the proposed approach outperforms the state-of-the-art approaches and delivers 6% to 27% improved

resource utilization estimation accuracy compared to baseline methods.

The work performed in this area has resulted in the following publication:

- Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, David Carrera, "Adaptive Prediction Models for Data Center Resources Utilization Estimation," IEEE Transactions on Network and Service Management

In this contribution we apply machine learning on telemetry monitored data focusing on adaptive ways to improve prediction accuracy, towards the next contribution where we focus on adaptive time window sizes for these machine learning models on-trend changing scenarios.

1.2.3 C3: Adaptive Window Size Selector for Prediction Models:

The **third contribution** of this thesis addresses the issue of observation window size, which is used to train the prediction model. The accurate size of observation windows helps in improving the prediction accuracy. Accurate prediction of data center resource utilization is required for capacity planning, job scheduling, energy saving, workload placement, and load balancing to utilize the resources efficiently. Existing prediction methods use fixed size observation windows, which cannot produce accurate results because of not being adaptively adjusted to capture local trends in the most recent data. Therefore, those methods train on large fixed sliding windows using an irrelevant large number of observations yielding to inaccurate estimations or fall for inaccuracy due to degradation of estimations with short windows on quick changing trends. In the third contribution, we propose a deep learning-based adaptive window size selection method, dynamically limiting the sliding window size to capture the trend for the latest resource utilization, then build an estimation model for each trend period. We evaluate the proposed method against multiple baseline and state-of-the-art methods, using real data-center workload data sets. The experimental evaluation shows that the proposed solution outperforms those state-of-the-art approaches and yields 9 to 40% improved prediction accuracy compared to the baseline methods.

The work performed in this area has resulted in the following submission:

- [Major revision submitted and under review] Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, David Carrera, "Adaptive Sliding Windows for Improved Estimation of Data Center Resource Utilization," Future Generation Computer Systems

1.3 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 introduces basic concepts which are used in completing the Ph.D. thesis goals. Chapter 3 introduces the telemetry reduction system based on Markov Chain Models. Chapter 4 presents the adaptive prediction model selection method to forecast the resource requirements for data centers accurately. Chapter 5 presents the adaptive training window size selection method to estimate the future need of resources in data centers and in last, Chapter 6 presents the conclusion and future work of this thesis.

Chapter 2

Background

This chapter introduces the relevant concepts which are later used in the following work. However, before discussing these concepts, we also include a brief discussion about the preliminary work of this thesis.

2.1 Telemetry Monitoring

The preliminary work consists of exploration and monitoring of telemetry streams for a single machine to characterize the workloads. For this purpose, we monitored the different type of telemetry data, including CPU, memory, context switches, I.O, memory bandwidth, interrupts, and others. This monitoring also helped us in evaluating the underlying hardware topology along with workload characterization. The details of this preliminary work are discussed in Appendix A.

2.2 Data center's Telemetry Monitoring

In the next phase, we move forward to data centers telemetry monitoring. Typical telemetry streams contain time series data about hardware utilization metrics including CPU, memory, I/O, bandwidth, context switches, interrupts, cache misses and cycles per instruction. These telemetry metrics are used in various applications. For example, CPU, memory, network bandwidth, instructions per cycle (IPC), cache miss rates, branch predictor statistics, and power consumption utilization data is used to forecast energy consumption and reduction [56, 34]. CPU, memory, disk, and network consumption are used for data center management and resource prediction [127, 117, 35, 100]. This monitoring generates a large amount of data due to which bandwidth utilization increases, and it also requires more storage space. The first contribution of the thesis is to evaluate the modeling techniques to reduce the bandwidth utilization and storage space requirements and for this purpose, we evaluated polynomial regression and markov chain models which are described in the following sections.

After exploring data reduction modeling techniques for data centers, we move forward to machine learning (ML) models. We used these ML models as a tool to predict future workload behaviors and traces. For this purpose, we use real traces of clusters from Alibaba, Materna and Bitbrains data centers and improve the prediction accuracy of CPU utilization. These data centers resource utilization exhibit variations and burstiness in their resource consumption due to dynamic nature of workloads. The second contribution is to propose a method which dynamically selects the most appropriate machine learning model according to recent resource consumption and for

this purpose, we evaluated different machine learning models such as linear regression, support vector regression, gaussian process regression, random decision forests and others. These are explained in the following sections. The next contribution of the thesis is to propose a method which selects the most appropriate window size for a machine learning model which is used to train it and for this purpose, we evaluated multi-layer perceptron (MLP) which is discussed in the last section of this chapter.

2.3 Markov Chains

Markov Chains are stochastic models describing a sequence of events in which the probability of each event depends only on the previous state of the event. Figure. 2.1 shows the irreducible ergodic two-state Markov Chain.

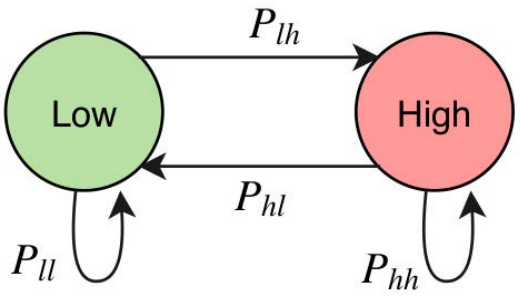


Figure 2.1: 2 states Markov Chain Model showing states and transition probabilities. For transition probabilities, l represents low state and h represents the high state.

Let $X_1, X_2, X_3, \cdots$ are independent and identically distributed random variables representing telemetry data. We model these random variables as a discrete time Markov Chain Model defining the probability of moving from the current state to the next state as:

$$Pr(X_{t+1} = s_j | X_t = s_i, ..., X_2 = s_2, X_1 = s_1) = Pr(X_{t+1} = s_j | X_t = s_i), \quad (2.1)$$

where function $Pr(X_{t+1} = s_j | X_t = s_i)$ is independent of t and denotes the probability of moving from state s_i at time t to state s_j at time $t+1$, symbol " | " represents the conditional probability, and $s \in$ state space (S).

Let P_{ij}^v , $v > 1$ where $P_{ij}^v = P(X_{u+v} = j | X_u = i)$ denotes the probability that after v time units the chain will transit to state j given that the current state is i at time u. The probability of reaching j from i in n-steps is the sum of all probabilities going from i to j through an intermediate point k. We use Chapman-Kolmogorov equation [115] to compute it as follows:

$$P_{ij}^{u+v} = \sum_{k \in S} P_{ik}^u P_{kj}^v; \quad m \text{ and } v \geq 1, \quad i \text{ and } j \in S. \quad (2.2)$$

Let $\mathbf{P}^v = (P_{ij}^v)$ be a matrix then Chapman-Kolomogorov equation can be expressed as $\mathbf{P}^{u+v} = \mathbf{P}^u \mathbf{P}^v$. This allows calculating the transition probability matrix P which reflects the relative frequencies of transitions from one state to another state. The

matrix P for n total number of states is represented as follows:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}. \tag{2.3}$$

2.4 Machine Learning

In this thesis, we use Machine Learning (ML) techniques for three main purposes: first, model data and represent it; second, predict future workload behaviors and traces; and third, from a set of ML methods and a context, choose one that predicts the workload better. To predict workload, we explore a diversity of Machine Learning techniques commonly used in the literature, ones more complex than others with different properties each. We explain each method in the following sections.

2.4.1 Linear Regression

Linear Regression (LR) is one of the simplest but effective approaches in machine learning modeling and prediction, specifically when a linear relation exists among variables. LR assumes there is a linear relation between output variable Y and input variables $X = \{x_1 \dots x_n\}$, and attempts to find a vector $W^T = \{w_1 \dots w_n\}$ and a scalar b where $\tilde{Y} = X \cdot W + b$ while minimizing the error $\epsilon = |Y - \tilde{Y}|$. Minimization is usually performed using the Least Squares Error approach, although other approaches using the deviation or specific cost function exist. LR variants include Polynomial and Multinomial Regression, where variable relations are assumed more complex, thus learning algorithms also become more complex.

2.4.2 Polynomial Regression

Polynomial regression is a form of regression in which the relationship between the dependent and the independent variable is modeled such that the dependent variable is an nth degree function of the independent variable. It is mostly used to fit the non-linear behavior between dependent and independent variable.

Mathematically, it is defined as

$$\mathrm{y} = \beta + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon \tag{2.4}$$

Where y is the dependent variable and the betas are the coefficient for different nth powers of the independent variable x starting from 0 to n. The calculation is often done in a matrix form, as shown below.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 \dots & x_1^m \\ 1 & x_2 & x_2^2 \dots & x_2^m \\ 1 & x_3 & x_3^2 \dots & x_3^m \\ \vdots & \vdots & & \\ 1 & x_n & x_n^2 \dots & x_n^m \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix} + \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{pmatrix} \tag{2.5}$$

2.4.3 Support Vector Regression

Support Vector Machine (SVM) methods are common for classification although they can be used for regression as Support Vector Regression machines (SVR) [32]. The

advantage of SVMs is that non-linear functions can be learned as linear ones thanks to a transformation of data known as the *kernel trick*.

SVMs allow learning non-linear functions by mapping them into a higher dimensional feature space, using a defined kernel function. Input X are mapped into an h-dimensional feature space using a predefined non-linear kernel function to produce a linear model. Similar to LR, we can express SVMs as $\tilde{Y} = k(X) \cdot W + b$, where k is the function making the space for X linear. SMVs error minimization consists of building two margin functions (support vectors) $X \cdot W + b \pm \epsilon$, where final error ξ is computed for those elements outside the margins. As a disadvantage, the margin ϵ can become an hyper-parameter.

2.4.4 Gradient Boosting

Gradient Boosting is the combination of the Gradient Descent optimization and Boosting techniques [77, 42]. As any other boosting technique, the learned model is the composition of weaker models focusing on subsets of data, forming a stronger model when combined. Usually, decision and regression trees are used on Gradient Boosting techniques, but any other modeling technique can be used for boosting.

On Gradient Boosting, a model is fitted as $\tilde{Y} = f(X)$ minimizing $\epsilon = |Y - \tilde{Y}|$. Then function f can be fine-grain tuned using another function h fitted to ϵ, learning and correcting the errors on the first function, and so on recursively. This recursion can continue until we rest satisfied with the resulting aggregation of models.

2.4.5 Gaussian Process Regression

Gaussian Process Regression (also known as *Kriging*) [55] is a non-parametric regression method, where the modeled function is trained after a Gaussian process using the covariances of previous examples. This process is used mainly for interpolation which requires some example observation points. Kriging method predicts by computing the weighted average of the values for neighbors from the known examples. Kriging models can model non-linear as well as linear behavior. Typical regression methods are extended by statistical models based on stochastic processes. However, Kriging also estimates the associated statistical variations using the distribution and correlation of observed data.

2.4.6 Ridge Regression (RR)

Ridge regression (RR) also known as Tikhonov regularization is improved version of LR by introducing regularization to constraining the coefficients to low range. This helps to reduce the chances of model over-fitting. The cost function for Ridge regression for hypothesis θ is calculated using:

$$\mathrm{R}(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T \cdot X^{(i)} - Y^{(i)})^2 + \alpha\frac{1}{2}\sum_{i=1}^{k}{\theta_i}^2, \tag{2.6}$$

where α is a hyperparameter use to control the regularization the model.

2.4.7 Least Absolute Shrinkage and Selection Operator (LASSO)

Least Absolute Shrinkage and Selection Operator (LASSO) is another improvement in LR by introducing regularization term and ensure to eliminates the least important

features to increase the model accuracy. The cost function for LASSO for hypothesis θ is calculated using:

$$\mathrm{L}(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T \cdot X^{(i)} - Y^{(i)})^2 + \alpha\sum_{i=1}^{k}\theta_i, \tag{2.7}$$

LASSO improves the prediction accuracy by selecting a subset of items rather than using all of them as compared to LR, NNLS, and Ridge regression which uses all of the features and data.

2.4.8 Elastic Net (EN)

EN improves LR using regulrization by combining Ridge and LASSO's regularizations. It also reduces the number of features by removing less important features to help in improving the accuracy of the model. The EN cost function is computed using:

$$\mathrm{E}(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T \cdot X^{(i)} - Y^{(i)})^2 + \frac{1-r}{2}\alpha\sum_{i=1}^{k}{\theta_i}^2 + r\alpha\sum_{i=1}^{k}\theta_i, \tag{2.8}$$

where r is a mix ratio and can be controlled to include regularization of Ridge and LASSO. For example, $r = 1$ will result the EN to behave similar to LASSO and $r = 0$ will force the EN to behave similarly to Ridge regression.

2.4.9 Non-Negative Least Square (NNLS)

NNLS is a type of constrained least problems to restrict the coefficients of the model to positive numbers. This type of regression methods is feasible for resource estimation as the output is always positive. The objective function of NNLS is:

$$\underset{\theta_j \geq 0 \quad \forall j}{arg\,min}\sum_{i=1}^{m}(\theta^T \cdot X^{(i)} - Y^{(i)})^2. \tag{2.9}$$

The objective function (2.9) ensures that the linear coefficients in θ are non-negative. Since the resource usage of data centers is always non-negative, therefore all values in Y are also non-negative. As a result, we get non-negative prediction. We used the algorithm proposed by Lawson and Hansonb [66] to solve the NNLS objective function.

2.4.10 K-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm allows to memorize a set of characteristic examples, and classify new data instances by finding the k *nearest neighbors*, and returning the class of the majority (or the probabilities per class on those k examples). The nearest neighbors are those examples with minimum distance, often euclidean, Hamming or Manhattan distances. Here we select k-NN as one of the tentative classifiers, as it is one of the easier models to train (it memorizes the training set), in exchange of the not-so-easy search process when predicting a new data instance.

2.4.11 Naïve Bayes

The Naïve Bayes algorithm is a classifier based on computing the likelihood of a feature given each class, then use the Bayes theorem to compute the conditional probability

of a class given that feature. The method extracts from data the probabilities of each feature value $P(Feature = X)$, each class $P(Class = C)$, and each likelihood of features per class $P(Feature = X|Class = C)$. This method assumes independence among features, in contrast to Bayesian Networks. The probabilities per class are the product of their probabilities per feature, and the algorithm returns the class with a higher probability (or the rank of classes per probability). We selected Naïve Bayes as one of the classifiers for its low complexity, as training implies keeping the count of element occurrences, then probabilities can be computed on demand at prediction time.

2.4.12 Random Decision Forest

Random Decision Forests (usually referred as Random Forests) are an ensemble method for classification and regression, based on the aggregation of specialized decision trees [51]. The ensemble builds a set of decision trees, trained from different data subsets, then predicted data is classified as the most voted class from all decision trees (the trend). The main reason to use random forests is to prevent over-fitting single decision tree models and get a more accurate and stable prediction. Random Forests are known to produce decent results for classification and regression problems, without the need for much tuning of hyper-parameters.

2.4.13 Multilayer Perceptron

Multilayer Perceptron (MLP) is a kind of Artificial Neural Network (ANN) used for both classification and regression problems for non-linear systems. The most commonly used ANN for classification problem is "one-hidden layer" Feed-Forward ANN, where the ANN is composed of a single layer of perceptrons (neuron units) and an output layer.

Data passes through the hidden layer to the output producing value for each class, then the class with a higher value is chosen. Neurons aggregate input data, usually through a linear function $X_o = X_i \cdot W + bias$, then passes outputs X_o to the next layer (here the output layer). Output neurons also pass their produced aggregation through sigmoid functions to approximate their outputs to 0 or 1 $Y = sigm(X_o)$. Fitting those functions is done by passing data repeatedly and comparing the network output with the real output, then updating neurons weights W and $bias$ using Gradient Descent techniques.

Neural networks can be complicated to fine-tune, as their architecture must be treated as a hyper-parameter, deciding how many neuron units are in the hidden layer, how many times data must be passed for training, etc.

Chapter 3

Telemetry Reduction by Markov Chain Models

This chapter presents the telemetry data reduction framework as the **first contribution** of this Ph.D. thesis. In this chapter, we address the collection and compression of large-scale data centers telemetry data. Our proposed solution consists of reducing the telemetry measurements of the data center in real time through online modeling using Markov Chains [57]. Then, such models are transmitted to the corresponding logging repositories and stored to enable data reconstruction for posterior use with minimum data loss to preserve the hardware utilization behaviors. The method works on rack level to collect all measurements from the nodes deployed on the rack and then applies Markov Chain Model to efficiently reduce the data in real-time. The reduced data is logged to allow telemetry data analytics. We also propose an efficient method to combine reduced data with compression to minimize the overall storage requirement for storing telemetry data for a long duration. Thus it minimizes both the storage space and the bandwidth utilization for collecting data center telemetry measurements.

The proposed method has been evaluated using real telemetry data sets and compared with state of the art methods such as the Polynomial Regression method [16] and the dictionary based compression (ZIP). Several comparison metrics were used, such as computing the amount of data saved in each scenario and the data reconstruction accuracy in the case of lossy compression. We evaluate the effectiveness of the methods on telemetry time-series data by calculating and comparing the storage requirements for each method. The reconstruction accuracy is evaluated by comparing data before compression and after reconstruction. First, we compare the statistical similarity between the reconstructed data and the original data using a two-sample Kolmogorov-Smirnov (KS) hypothesis tests [61]. KS test is used to identify whether two given one-dimensional sequences belong to the same probability distribution or not. It does not quantify the similarity of the reconstructed data but indicates whether the reconstruction has a statistical resemblance. Then we complement the evaluation by quantifying this similarity using the Dynamic Time Wrapping (DTW) metric [103, 81]. The DTW is a well-known method used to measure the similarity between two given sequences which may vary in speed [50, 81, 8, 98, 86].

The main contributions of this chapter are summarized as follows:

- Design a system for real-time telemetry data reduction and reconstruction for data centers.
- Develop and evaluate telemetry data reduction and reconstruction approach using Markov Chain models.

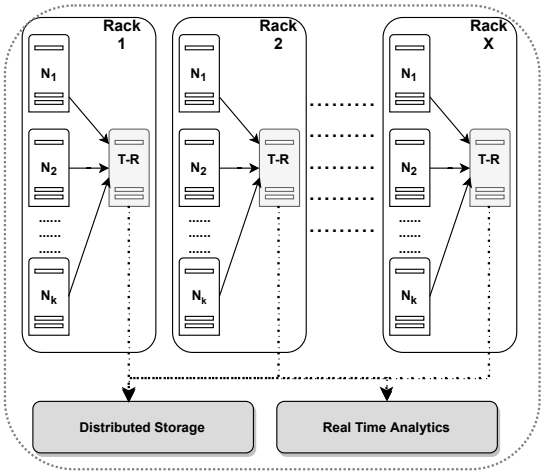


Figure 3.1: Schema of the proposed data reduction framework for data centers. Each rack in the data center hosts a Telemetry Reductor (T-R) component which continuously collects the telemetry data from each computing node, then performs data reduction before transmitting it to storage and real-time analytic systems.

- Compare the proposed data reduction and reconstruction with state-of-the-art Polynomial Regression based methods and ZIP compression.
- Experimental evaluation to study the storage and bandwidth minimization using the proposed solution for telemetry data for different data center sizes.

3.1 Data Reduction Framework

Our proposed system to reduce telemetry data provides two-fold benefits: first, it reduces the storage space significantly, and second, it minimizes the bandwidth utilization required by telemetry data collection within the data center.

Figure 3.1 shows the architecture of the proposed real-time telemetry data reduction system for data centers. Each rack in the data center hosts a Telemetry Reductor component which continuously collects the telemetry data from each computing node and performs data reduction method before transmitting it to storage and real-time analytic systems. This reduces the data at the rack level and does not transmit the entire data but only the reduced data within the data center. A telemetry stream data can consist of utilization of CPU, memory, disk, network, memory bandwidth and other useful metrics.

Figure 3.2 shows the Telemetry Reductor process. We used Apache Kafka [47, 62] as a message broker to receive telemetry stream data from the computing nodes. Every computing node publishes telemetry data with the timestamp to Kafka topic which is consumed by the Telemetry Reductor. Each consumer obtains data from Kafka topics. The consumer reads the incoming streams and splits the data into a predefined batch size and uses a data reduction method to minimize the data. Then the reduced data is compressed and stored in a data center storage facility. The reduced data is also fed to analytics engines. The telemetry data can be used for real-time monitoring, workload characterization, and anomaly detection purposes. The reduced data sent to other components need to be reconstructed before usage. The above architecture is based on out-of-band monitoring where data is coming from devices such as sensors or

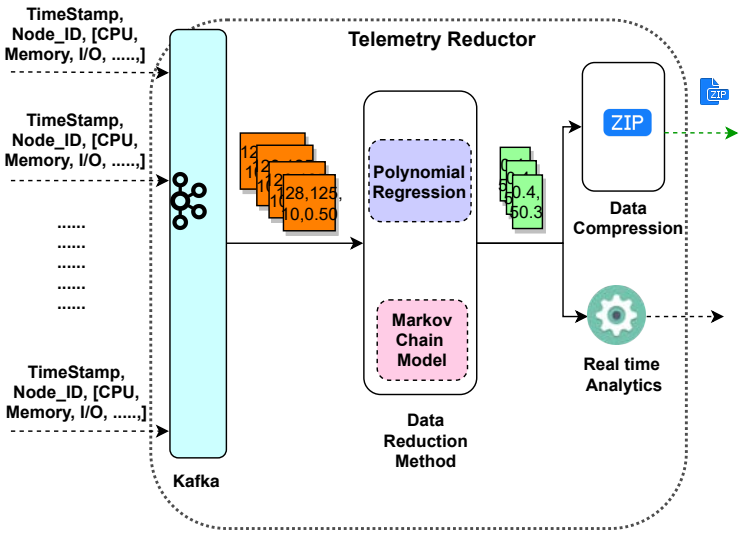


FIGURE 3.2: Telemetry Reductor process the incoming telemetry streams from computing nodes using Kafka and data reduction method. The output of the Telemetry Reductor can be used to store or provide input to other real-time analytics systems.

logging applications. In case of the regular host where in-band monitoring is possible then Telemetry Reductor component could be executed on the host machine without Kafka to reduce the extra overhead of a dedicated machine.

The proposed framework targets two main goals: i) it minimize the storage demand for storing telemetry measurement by reducing and compressing the telemetry data. ii) It minimizes the bandwidth utilization by transmitting solely reduced data over the data center network. To achieve such goals, we propose two different methods namely Polynomial Regression and Markov Chain models to be used for telemetry data reduction and reconstruction in real-time. We explain both of these methods in the chapter 2.

3.1.1 Reduction through Polynomial Regression (PR) methods

We use Polynomial Regression (PR) methods to fit the curve of a given telemetry data-stream into a polynomial curve. Then we only need to store the coefficients of the equation fitting the curve. This method is inspired by similar work [16] that uses linear regression method for data reduction. The PR method is used as a baseline to compare our proposed Markov Chain model-based approach. To understand the effect of using PRs for data reduction, if we plan to fit a data-stream function into a 4-degree polynomial curve then we will only store 4 coefficients plus the intercept. Assuming that the data-stream contains 128 data points, we will be reducing the data from 128 values to only 6 values.

If we train a PR model for each telemetry measurement, given a n-degree polynomial regression, and k observations of a specific telemetry dimension a, the coefficients are computed using pseudo-inverse solution to minimize the sum of least squares:

$$\begin{pmatrix} \alpha_o \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \left[R^\top R\right]^{-1} R^\top \begin{pmatrix} r_t \\ r_{t-1} \\ \vdots \\ r_{t-k} \end{pmatrix} \tag{3.1}$$

where the matrix R is defined as

$$R = \begin{pmatrix} 1 & a_t & a_t^2 \dots & a_t^n \\ 1 & a_{t-1} & a_{t-1}^2 \dots & a_{t-1}^n \\ \vdots & \vdots & & \\ 1 & a_{t-k} & a_{t-k}^2 \dots & a_{t-k}^n \end{pmatrix}. \tag{3.2}$$

Using the values of $\alpha_0, \alpha_1, \cdots, \alpha_n$ coefficients and polynomial degree n, we can load the polynomial curve and reconstruct the k data points easily. Therefore, we propose to only store coefficients and polynomial degree information instead of the actual data. Here we monitor and collect the data for a given batch interval and then fit a PR model with a specific degree. After fitting the model, we extract the coefficients from the fitted equation and store these instead of the actual data. Whenever we have to reconstruct a batch of data, we load the coefficient values for that batch and build its corresponding PR model, and then we use it to reconstruct the data points for the batch.

3.1.2 Reduction through Markov Chain Models (MM)

Secondly, we propose to use Markov Chain Models (MM) [57] for telemetry data reduction and reconstruction. Markov Chains are stochastic models describing a sequence of events in which the probability of each event depends only on the previous state of the event. Specifically, we use time-homogeneous discrete time Markov Chains (DTMC) [74] because telemetry data is monitored at discrete time intervals and the state transition probabilities are independent of time. Moreover, the DTMC used with the telemetry data is irreducible ergodic as the proposed system can transit from every state to every other state with positive probability. Figure. 3.3 shows the irreducible ergodic 2 state Markov Chain.

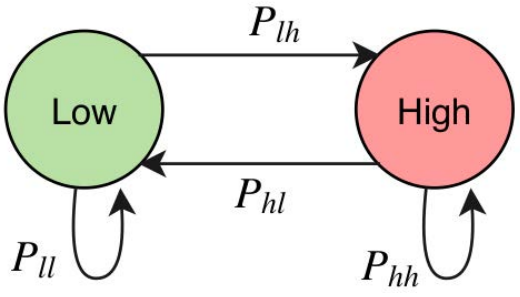


Figure 3.3: 2 states Markov Chain Model showing states and transition probabilities. For transition probabilities, l represents low state and h represents high state.

The idea of using MMs is to explore that if we can deal with burstiness behaviors among others for data reduction and reconstruction. The burstiness behavior represents sudden spikes and peaks in the telemetry data. In general, it is challenging to reconstruct burstiness behavior, and we address it by using Markov Chain Models.

Let $X_1, X_2, X_3, \cdots$ are independent and identically distributed random variables representing telemetry data. We model these random variables as a discrete time Markov Chain Model defining the probability of moving from the current state to the next state as:

$$Pr(X_{t+1} = s_j | X_t = s_i, ..., X_2 = s_2, X_1 = s_1) = Pr(X_{t+1} = s_j | X_t = s_i), \quad (3.3)$$

where function $Pr(X_{t+1} = s_j | X_t = s_i)$ is independent of t and denotes the probability of moving from state s_i at time t to state s_j at time $t+1$, symbol " | " represents the conditional probability, and $s \in$ state space (S).

Let P_{ij}^v , $v > 1$ where $P_{ij}^v = P(X_{u+v} = j | X_u = i)$ denotes the probability that after v time units the chain will transit to state j given that the current state is i at time u. The probability of reaching j from i in n-steps is the sum of all probabilities going from i to j through an intermediate point k. We use Chapman-Kolmogorov equation [115] to compute it as follows:

$$P_{ij}^{u+v} = \sum_{k \in S} P_{ik}^u P_{kj}^v; \quad m \text{ and } v \geq 1, \quad i \text{ and } j \in S. \quad (3.4)$$

Let $\mathbf{P}^v = (P_{ij}^v)$ be a matrix then Chapman-Kolomogorov equation can be expressed as $\mathbf{P}^{u+v} = \mathbf{P}^u \mathbf{P}^v$. This allows calculating the transition probability matrix P which reflects the relative frequencies of transitions from one state to another state. The matrix P for n total number of states is represented as follows:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}. \quad (3.5)$$

In our solution, the chain can transit from every state to every other state and considered as irreducible ergodic Markov chain. For an irreducible ergodic Markov chain, the transition matrix elements must be non-negative, $p_{ij} \geq 0$ and the sum of each row must be equal to 1, therefore, $\sum_{j=1}^{n} p_{ij} = 1$.

To reduce a given telemetry data, first we convert it into state interval matrix $I = [...]_{(n+1)\times 1}$ and then compute state transition frequency matrix $F = [...]_{n \times n}$, where n is the total number of states. The state interval matrix contains the threshold values to partition the given data into n states. The state transition frequency matrix contains the transition frequencies which are computed by simply counting the number of transitions from one state to another.

Figure 3.4 shows the process to compute the frequency matrix for reducing telemetry data using a 2 states MM with batch size 16. First, we calculate the state interval matrix I which stores the partition boundaries. For 2 states MM, the state interval matrix will be of size 3×1 to store the boundaries of the states. Second, we convert each data point into corresponding states using the state interval matrix. Finally, we compute the state transition frequencies matrix F. We only store F and I for each given batch of data to reduce the telemetry data. During data reduction, we do not compute and save the transition probability matrix P because it is required during the reconstruction and can be easily calculated using F. Moreover, P consists of floating point data which also introduces storage overhead.

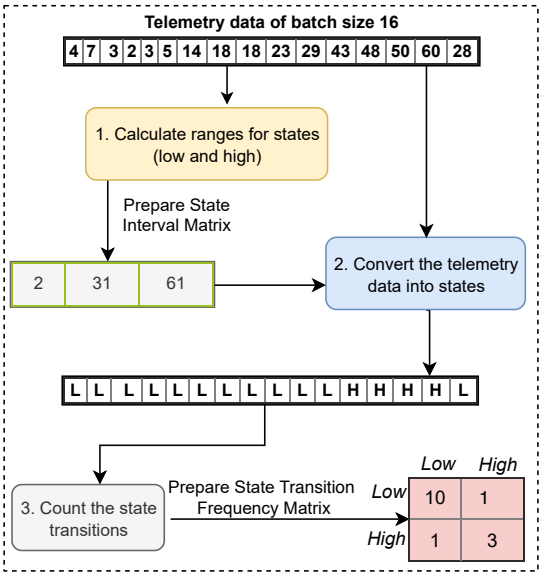


Figure 3.4: Telemetry data reduction using 2 state Markov Chain Model and batch size 16.

To reconstruct the data for a given batch interval, we first convert the state transition frequency matrix F into state transition probability matrix P using the equation 3.6.

$$p_{ij} = \begin{cases} \frac{f_{ij}}{\sum_{r=1}^{x} f_{ir}}, & \text{if} \quad \sum_{r=1}^{x} f_{ir} \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

During the reconstruction of a batch data, we assume that the current/initial state is known and we need to predict all data points of the given batch size. To explain the reconstruction strategy, let us consider a 2 states MM as shown in Figure 3.3. The transition probabilities P can be easily computed from the corresponding transition frequency matrix F which was stored during the data reduction phase. Let us assume that the current state is *high*, the transition probability from *high* to *low* (p_{hl}) is 0.3 and the transition probability of *high* to *high* (p_{hh}) is 0.7. We generate a random number between 0 and 1. If the generated number is greater than 0.3 then we predict *high* otherwise *low* as the next state. Once we identify the next state, then we look up the lower and boundary values of the predicted state from the state interval matrix I. Finally, we generated another random number within the range of the state boundary and considered it as the reconstructed data point. We repeat this process to identify all data points for the given batch interval.

3.2 Experimental Setup

3.2.1 Description of the Data-set

We used IBM POWER8 telemetry logs data set [4] to evaluate our proposed method for telemetry reduction and reconstruction. These logs contain telemetry data generated by executing three representative Spark workloads from the Spark-Bench [109] developed by the IBM and widely tested using POWER8 systems. The data set logs are collected from executions of the workloads "Support Vector Machines (SVM)", "PageRank" and "Spark SQL". These workloads are well known in the literature and

combine different characteristics to cover a large range of different resource usage behaviors. The data set contains metrics related to CPU, memory, context switches, memory bandwidth, L2 and L3 cache misses, interrupts, and cycles per instruction (CPI) as a time series data.

3.2.2 Experiment Details

As explained previously, we propose and evaluate two different techniques, namely Polynomial Regression and Markov Chain Models, to reduce and regenerate telemetry data through modeling. To evaluate the proposed methods, we performed two major experiments, briefly explained in the following subsections.

Experiment 1: Data Reduction and Reconstruction using Polynomial Regression

In this experiment, we study the effect of different polynomial degrees and batch sizes on PR models. The polynomial degree defines the shape of the curve. Where a higher degree can be used to fit a complex curve and a lower degree can be used a simple curve. The batch size defines the number of data points used to fit the curve. We consider polynomial degrees 2, 4, 6, 8, and 10 with batch sizes varying from 2, 4, 8, 16, 32, 64, and 128 to fit polynomial curves for data reduction and reconstruction. Many other settings can also be studied, however, the selected settings are sufficient to establish the motivation for using PR for telemetry data reduction and reconstruction because with higher degrees, we get less reduction and with larger batch sizes, we lose the accuracy of the reconstructed data. Once we train a PR model using a specific polynomial degree and a batch size then we only store the coefficients of polynomial equation learned to fit on the given data points. This helps to reduce the data size significantly as we do not store all data point but only few coefficient values. Later, these coefficients can be used to regenerate the data points easily. However, an efficient polynomial degree and batch size values should be used to achieve a good data reconstruction accuracy: the higher the degree, the more over-fitting will be achieved, but more data (coefficients) will be transmitted.

Experiment 2: Data Reduction and Reconstruction using Markov Models

In this experiment, we study the effect of different MM models with varying the number of states and batch sizes. We consider 2, 3, and 4 state Markov Models on batch sizes varying from 2, 4, 8, 16, 32, 64, and 128 to study the effects on data reduction and reconstruction. Larger batch sizes reduce accuracy as the reconstruction does not retain the information which is present in the original data set. On the other hand, a higher MM model states yield less reduction. Figure 3.3 shows a 2 state Markov model in which we divided the input data into two regions namely low and high and learn a state transition matrix for telemetry observations using a specific batch size. The state transition matrix contains the probabilities of moving from one state to other using the given input data points of the telemetry metrics. We also create a state interval matrix which defines the low and high region ranges. These two matrices are learned and stored instead of the complete data which reduces the data size significantly. Similarly, for 3 and 4 state Markov Models, we increase the number of states to 3 and 4 respectively and learn state transition metrics and state interval matrix accordingly.

3.2.3 Evaluation Criteria

We evaluate both of the proposed methods in terms of data reduction effectiveness and reconstruction accuracy. For evaluating the effectiveness of data reduction, we calculate the data reduction percentage after applying the proposed methods. For reconstruction accuracy, first, we perform two-sample Kolmogorov-Smirnov (K-S) test to decide whether to accept or reject the produced reconstruction of data after compressing, storing and reconstructing it. This first evaluation is used to initially discard configurations (PR degrees and MM discrete states) that produce low quality reconstructions. Second, we compute Dynamic Time Warping (DTW) distance between reconstructed data and actual data to quantify the reconstruction error. We explain all these evaluation measures in the following subsections.

Data Reduction Percentage

We compute the data reduction percentage by measuring the data stored after applying the reduction method against the original data. A positive value shows a reduction in data size while a negative value indicates growth in data size. Therefore, a higher data reduction percentage is better and desirable.

3.2.4 Two-sample Kolmogorov-Smirnov (K-S) Test

We use two-sample Kolmogorov-Smirnov (K-S) test [61] to compare the statistical similarity of the actual data with the reconstructed data. The K-S test is used to determine whether two given one-dimensional sequences belong to the same probability distribution or not. The output of the K-S test is a p-value. A p-value lower than or equal to 0.025 indicates that the given two sequences are not drawn from the same probability distribution. However, a p-value higher than 0.025 indicates that the given two sequences are statistically similar [87, 41]. Therefore, in our evaluation, we divided the p-value into two regions namely *Accepted (A)* when the p-value is greater than 0.025 and *Not Accepted (NA)* when the p-value is less than or equal to 0.025.

3.2.5 Dynamic Time Warping (DTW)

The K-S test can be used to see whether the generated sequence is statistically comparable to the original one, but it does not quantify the sequential similarity of the reconstructed data. Therefore, to quantify the error against the actual data, we used the Dynamic Time Wrapping (DTW) distance metric [103, 81]. This is a well-known method used to measure the similarity between two given sequences which may vary in speed [50, 81, 8, 98, 86]. A small value of the DTW test is considered good as it shows that the given two sequences are close to each other. However, a large value of the DTW test is considered bad as it indicates that the two given sequences are not close to each other. Therefore, a lower value of DTW is desirable to consider the reconstructed data similar to the actual data.

3.3 Experimental Results

In order to study and validate the presented methods, we compare both techniques (PRs and MMs), using the described data sets, by evaluating the aforementioned metrics (K-S, DTW and reduction improvement). Also, we compared our approach against directly applying classical data compression mechanisms.

Table 3.1: Average p-values of two-sample K-S test using PR with different degrees and batch sizes for user CPU, memory free, context switches, and memory bandwidth hardware metrics in Experiment 1. The Not Accepted values are denoted by the red line.

user cpu						context switches					
BS	**PR2**	**PR4**	**PR6**	**Pr8**	**PR10**	**BS**	**PR2**	**PR4**	**PR6**	**Pr8**	**PR10**
2	0.945	0.945	0.945	0.945	0.945	**2**	1	1	1	1	1
4	0.917	0.943	0.943	0.943	0.943	**4**	0.934	1	1	1	1
8	0.645	0.769	0.821	0.758	0.758	**8**	0.679	0.864	0.953	0.999	0.999
16	0.397	0.550	0.640	0.686	0.718	**16**	0.309	0.507	0.646	0.750	0.836
32	0.125	0.212	0.268	0.298	0.324	**32**	0.075	0.147	0.204	0.241	0.269
64	0.043	0.080	0.102	0.134	0.168	**64**	~~0.011~~	0.027	0.045	0.067	0.096
128	~~0.005~~	~~0.013~~	~~0.024~~	0.033	0.042	**128**	~~0.004~~	~~0.007~~	~~0.016~~	~~0.010~~	~~0.010~~
memory free						**memory bandwidth**					
BS	**PR2**	**PR4**	**PR6**	**Pr8**	**PR10**	**BS**	**PR2**	**PR4**	**PR6**	**Pr8**	**PR10**
2	1	1	1	1	1	**2**	1	1	1	1	1
4	1	1	1	1	1	**4**	0.989	1	1	1	1
8	0.991	0.998	0.999	1	1	**8**	0.826	0.914	0.974	0.988	0.988
16	0.924	0.952	0.967	0.974	0.976	**16**	0.500	0.684	0.831	0.873	0.901
32	0.725	0.814	0.849	0.857	0.870	**32**	0.210	0.389	0.473	0.515	0.544
64	0.468	0.555	0.637	0.695	0.725	**64**	0.063	0.132	0.215	0.297	0.371
128	~~0.217~~	0.293	0.363	0.404	0.429	**128**	~~0.012~~	~~0.022~~	0.047	0.075	0.103

3.3.1 Experiment 1: Data Reduction and Reconstruction Using Polynomial Regression (PR)

In this first experiment, we calculated the data reduction obtained from using PR models with different data batch sizes and different polynomial degrees. We then evaluated the results using a two-sample K-S test to discard inappropriate solutions. Finally, we computed the DTW distance to quantify the quality of the solution. This evaluation is presented in the following subsections.

Two-sample K-S Test

Table 3.1 shows the results of a two-sample K-S test. First, we reconstructed the data using the corresponding coefficients, intercept, polynomial degree, and batch sizes for each PR model stored during the data reduction phase. Then, we apply the two-sample K-S test to compute the average p-value for each telemetry measure. A p-value lower than 0.025 is considered bad and does not reflect appropriate similarity with the actual data. However, a p-value higher than 0.025 is considered acceptable and reflecting the statistical feasibility of reconstructed data belonging to the same distribution as the actual data [87, 41]. The *Not Accepted (NA)* values are denoted by the red line in the table.

Most of the polynomial regression models reconstructed the data within an acceptable range. For example, for user CPU metric, only polynomial degree 2, 4, and 6 using batch size 128 reconstructed the data with not acceptable range, while all other PR models yielded an acceptable range. In the case of *context switch* metric, all PR models with batch size 128 reconstructed the data within not acceptable range, the same was for PR2 with batch size 64 while all remaining PR models reconstructed data within an acceptable range.

Data Reduction Percentage

Table 3.2 shows the data reduction percentage using PR models learned for different polynomial degrees and batch sizes for Experiment 1. The negative values in the table

Table 3.2: Data reduction percentage using PR with different polynomial degrees and batch sizes for Experiment 1. The red line values are Not Accepted and taken from table 3.1. The grey shaded negative values represent data growth.

Batch Size	PR2	PR4	PR6	PR8	PR10
2	-42.86	-60.36	-77.66	-94.96	-129.43
4	-3.58	-43.35	-51.82	-60.51	-77.89
8	39.21	-3.15	-47.72	-69.19	-78.22
16	68.06	46.78	25.54	4.39	-18.92
32	82.95	75.64	69.69	65.95	60.43
64	~~90.08~~	84.90	78.62	74.28	67.39
128	~~93.08~~	~~91.80~~	~~90.11~~	~~87.18~~	~~86.26~~

show growth in data size instead of reduction. The negative values are observed due to two reasons. First, whenever the batch size (i.e., the number of data points) is smaller or equal to the polynomial degree used to fit the curve, it results in learning more coefficients than the actual data points. Hence, the data size grows compared to the original data. Second, sometimes the PR models coefficients consist of high precision decimal values which required more spaces compared to the actual data points. A higher polynomial degree model is very sensitive to the coefficient values and rounding the coefficient values significantly changes the shape of the curve.

To understand the growth of data using PR models, consider PR6 with batch size 2 which increases the data by 77.66%. This is mainly due to the fact that for every batch interval we need to store six coefficients and one intercept (seven data points in total) while the actual data consists of only two data points. Therefore, we should avoid fitting a curve on a batch size smaller than the polynomial degree used to fit the data. Another case is to consider PR6 with batch size 8 in which data size increase by 47.74% mainly due to the high precision of the coefficients.

We observed that PR models with large batch size and small polynomial degrees help in reducing the telemetry data significantly. For example, batch size 64 with PR4 reduces data to 84.90%. However, such PR models may not regenerate the data with good accuracy, specifically for bursty and noisy telemetry observations. Therefore, we need to identify an appropriate combination of batch size and the polynomial degree to reduce the data size with higher data reconstruction accuracy.

DTW Distance

Figure 3.5 shows the normalized average DTW distance for PR models learned using different degrees and batch sizes for Experiment 1. The batch sizes without reduction are denoted by NR and batch sizes which are in not acceptable regions are denoted by NA. We observed that on large batch sizes the DTW distance increases and it decreases by increasing the PR degree. We observed that on batch size 8, only degree 2 (PR2) is reducing the data with 98% accuracy. The accuracy of reconstructed data at batch size 16 varies from 95% to 98% where it reduces to 70 to 80% at batch size 64.

Figure 3.6 shows the reconstruction of first 300 data points of PageRank workload of our data set using different regression models trained on degree 2 with a lower batch size (BS 8) and on degree 10 with a higher batch size (BS 64). We observed that BS 8 with degree 2 yields better reconstruction even for spikes and burstiness comparing to BS 64 but it only achieved 39.21% data reduction. However, BS 64 with

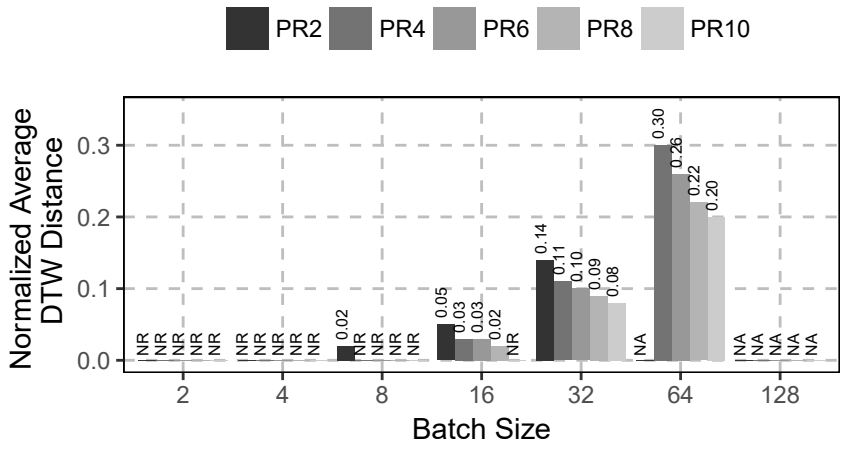


Figure 3.5: Normalized average DTW distance for telemetry measures using PR models trained on different batch sizes and polynomial degrees for Experiment 1. (NR = Not Reduced, NA = Not Accepted)

degree 10 yields 67.39% data reduction, the K-S test is also acceptable but the DTW distance for BS 64 model is higher than BS 8 model for all telemetry metrics. This confirms that PR with a higher degree and larger batch size yields less accuracy in reconstructed data.

3.3.2 Experiment 2: Data Reduction and Reconstruction Using Markov Model (MM)

In this second experiment, we calculated the data reduction obtained from using Markov Models with different data batch sizes and a different number of Markov states. We then evaluated the results using a two-sample K-S test to discard inappropriate solutions. Finally, we computed the DTW distance to quantify the quality of the solution. The following subsections present the evaluation results.

Two-sample K-S Test

Table 3.3 shows the results of the two-sample K-S test. Most of the Markov Models reconstructed data are within the acceptable range except few such the reconstructed user CPU metrics using the 2-state MM with a batch size of 128 are not within the acceptable range. The reconstructed memory free metrics using MM models with batch size 128 are not within the acceptable range. The results conclude the fact that all reconstructed telemetry metrics are within the acceptable range when using Markov Model with 2, 3 and 4 states and up to 64 batch size.

Data Reduction Percentage

Table 3.4 shows the data reduction percentage for different batch sizes with 2, 3 and 4 state MM. The negative values in the table show a data growth instead of reduction. The negative values are observed whenever the number of data points learned as part of MM model turns higher than actual data points.

We observed that large batch sizes and small state value yield higher data reduction. For example, batch size 64 with 2 states yields 92.81% data reduction. For

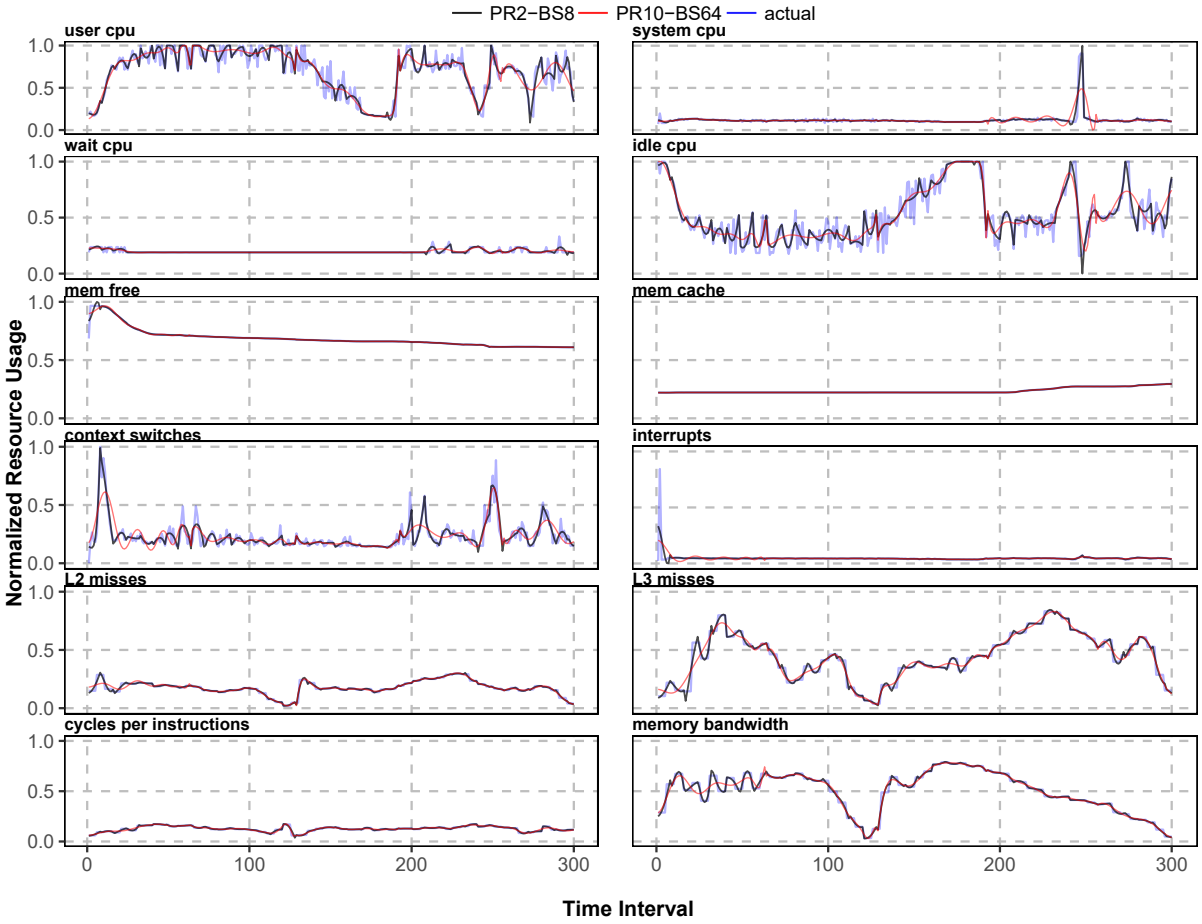


Figure 3.6: Reconstruction of PageRank workload telemetry data using polynomial degree 2 with batch size 8 (PR2-BS8) and polynomial degree 10 with batch size 64 (PR10-BS64).

2 and 3 states MM, we start observing data reduction after batch size greater than 2. However, for 4 state MM, we observe data reduction for batch sizes greater than 4. Hence, the data reduction depends on the size of the transition probability matrix and the state interval matrix. For example, we need 2 *times* 2 matrix to store state transition probabilities and 3 data points to represent the state interval matrix. Therefore, for 2 state MM, we need at least 7 data points to represent a given batch. We observed 39% data reduction on batch size 4 when the original data points are 4 and MM data points are 7. The reason of this reduction is that some of the telemetry metrics have at maximum 9 digits in their actual data e.g context switches, interrupts whereas state transition probability matrix contains only counts of moving from one state to another. Thus it requires less storage even if MM data points are higher than the original batch size. This count is later converted into probabilities whenever reconstruction is required.

DTW Distance

Figure 3.7 shows the normalized average DTW distance for Markov Models learned using different degrees and batch sizes for Experiment 2. The batch sizes without reduction are denoted by NR and batch sizes which are in not acceptable regions are

Table 3.3: Average p-values of two-sample K-S test using Markov Models (MM) with different states and batch sizes for user CPU, memory free, context switches, and memory bandwidth hardware metrics in Experiment 2. The Not Accepted (NA) values are denoted by red line.

user cpu				context switches			
BS	**2MM**	**3MM**	**4MM**	**BS**	**2MM**	**3MM**	**4MM**
2	0.999	0.999	0.999	**2**	0.996	0.996	0.996
4	0.663	0.674	0.663	**4**	0.745	0.766	0.775
8	0.403	0.401	0.408	**8**	0.579	0.622	0.656
16	0.281	0.285	0.294	**16**	0.408	0.485	0.543
32	0.181	0.203	0.215	**32**	0.237	0.331	0.396
64	0.076	0.113	0.126	**64**	0.107	0.189	0.246
128	~~0.023~~	0.037	0.060	**128**	~~0.013~~	0.053	0.092
memory free				**memory bandwidth**			
BS	**2MM**	**3MM**	**4MM**	**BS**	**2MM**	**3MM**	**4MM**
2	1	1	1	**2**	1	1	1
4	1	1	1	**4**	0.800	0.843	0.892
8	0.998	0.998	0.998	**8**	0.271	0.328	0.354
16	0.937	0.927	0.906	**16**	0.115	0.139	0.129
32	0.658	0.580	0.545	**32**	0.082	0.068	0.072
64	0.124	0.088	0.063	**64**	0.057	0.053	0.051
128	~~0.002~~	~~0.003~~	~~0.004~~	**128**	0.026	0.028	0.034

Table 3.4: Data reduction percentage using Markov Models (MM) with different states and batch sizes. The red line values are Not Accepted and taken from table 3.3. The grey shaded negative values represent data growth.

Batch Size	2MM	3MM	4MM
2	-30.15	-106.06	-198.15
4	39.00	2.38	-42.06
8	70.05	53.08	31.50
16	83.09	75.13	63.77
32	90.28	84.64	79.38
64	92.81	91.30	87.30
128	~~93.95~~	~~93.16~~	~~92.16~~

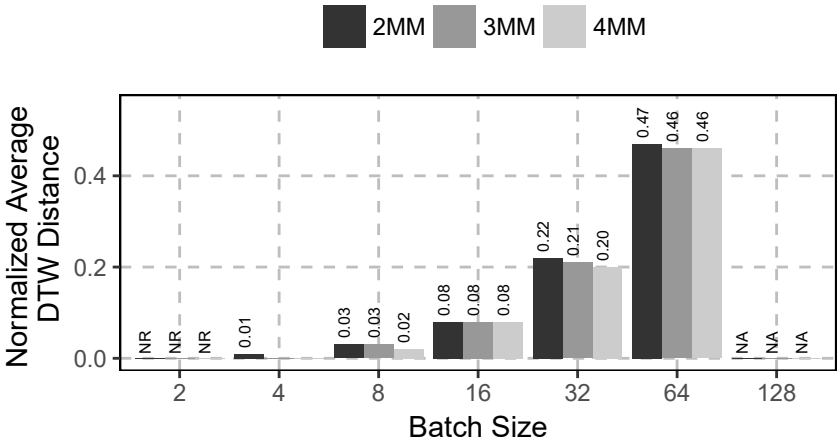


Figure 3.7: Normalized average DTW distance for MM trained on different batch sizes and states for telemetry measures. (NR = Not Reduced, NA = Not Accepted)

denoted by NA. We observed that on large batch sizes the DTW distance increases. We also observed that on batch size 4, only 2-state MM model (2MM) is reducing the data with 99% accuracy. The accuracy of reconstructed data at batch size 8 varies from 97 to 98% and at batch size 16, it is 92% where it reduces to 53 to 54% at batch size 64. The MM models do not show any effect on telemetry measurements which do not contain spikes, bursts, or noise. For example, free memory (MEM free) telemetry measurement of our data set does not have any effect on DTW using different MM models trained on different MM states and batch sizes. Therefore, such type of telemetry measurement can be reconstructed using a small number of MM states.

Figure 3.8 shows the reconstruction of first 300 data points of PageRank workload of our data set using different Markov Models trained using 2 states and a batch size of 8 (2MM-BS8) and 4 states and a batch size of 64 (4MM-BS64). We observed that BS 8 with 2 state MM reconstructed the data appropriately. However, with 4 states model with batch size 64 does not show good reconstruction. The data reduction using 2MM-BS8 model only reduces 70.05% data while 4MM-BS64 reduces 87.30% of the data. The K-S test is also acceptable for both of these models while the DTW distance for the BS64 model is higher than the BS8 for all telemetry metrics.

Using a small batch size (BS8) helps to capture the data patterns well including spikes and burstiness compared to a large batch size (BS64). It shows that spikes and noise in the original data cannot be well captured using larger batch sizes. For example, system CPU had few spikes around 270 seconds in the actual data. These spikes are well captured using batch size 8, however batch size 64 does not capture these spikes well. The reason for this behavior is mainly due to the fact that MM depends on the state transition matrix which contains the probabilities of moving from one state to another. Therefore, if there is burstiness in a specific batch then the probability of having burstiness remains for the whole batch interval. Thus the performance of the MM method in data reconstruction is not always robust using large batch size. We conclude that, if data contains spikes and burstiness then we should use smaller batch sizes with MM method. However, if we can detect that data does not contain spikes and burstiness then we can use higher batch sizes with MM method. This behavior is well observed in memory free and memory cache telemetry metrics presented in Figure 3.8.

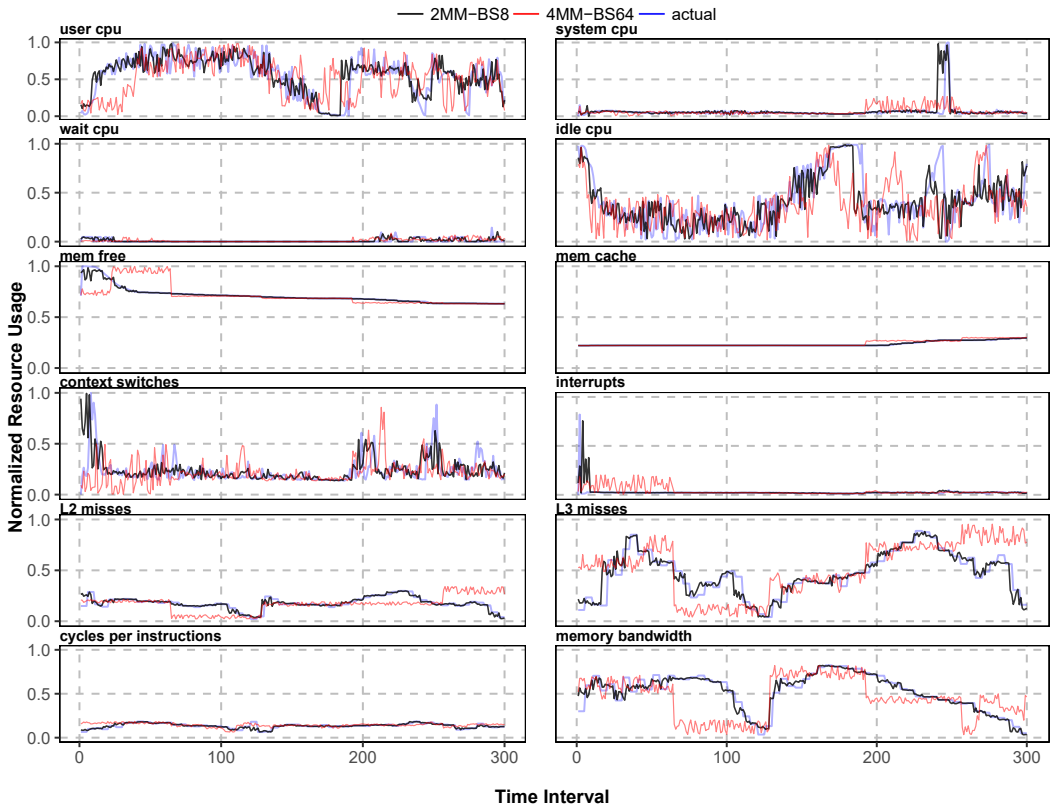


Figure 3.8: Reconstruction of PageRank workload telemetry data using 2 state Markov model with batch size 8 (2MM-BS8) and 4 state Markov Model with batch size 64 (4MM-BS64).

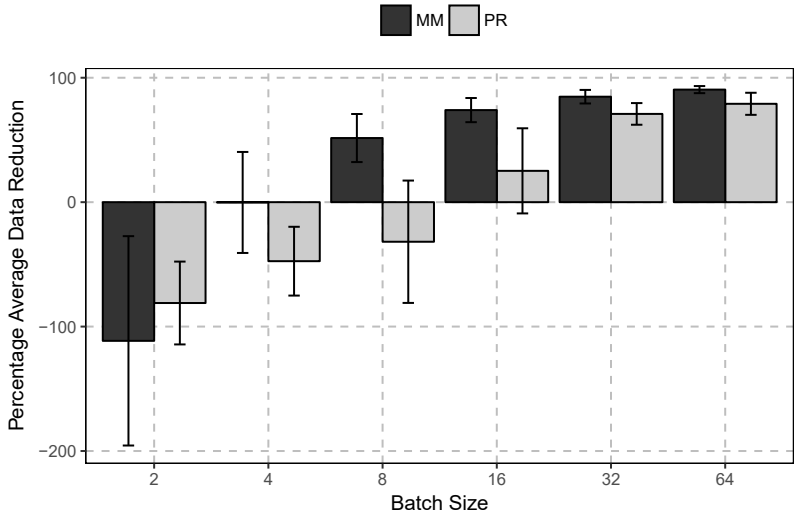


Figure 3.9: Comparison of average data reduction percentage using Polynomial Regression (PR) and Markov Models (MM) methods for telemetry data reduction.

3.3.3 PR and MM Comparison

To compare the PR and MM methods, we compute the average percentage data reduction and the average DTW distance for all settings (polynomial degrees and states) of these methods on different batch sizes. In this section, we show the comparison of PR and MM methods for data reduction and reconstruction.

Figure 3.9 shows the average data reduction on different batch sizes for all settings using PR and MM methods. In average, the batch sizes less than 8 do not yield any data reduction but instead they cause data growth. However, a significant gain of 51.54% is observed using MM method on batch size 8 compared to 31.81% for the PR method. For batch sizes higher than 8, MM always outperforms PR in data reduction.

Figure 3.10 shows the average normalized DTW distance for PR and MM method on different batch sizes. In average, the batch sizes less than 16 yields very low DTW distance, less than 0.2, for both PR and MM methods which reflects a good similarity of the reconstructed data with the actual data. For large batch sizes e.g., 32 and 64 the PR method outperforms MM in data reconstruction.

Large batch sizes reduce data significantly, however, they perform poorly in data reconstruction similarity. Therefore, from Figures 3.10 and 3.9, we conclude that batch size 16 is appropriate to use with both PR and MM methods because we obtain 25.17% and 74% average data reduction for both PR and MM respectively with DTW distance below 0.2 for both methods. However, we prefer to use MM model mainly due to higher data reduction on batch size 16 although PR method has slightly better DTW.

3.3.4 Data Reduction Using ZIP Compression

After comparing the two proposed methods, we evaluate them against the usage of classic lossless compression methods such as ZIP algorithms. We compared the ZIP algorithm on raw data with applying the ZIP algorithm on the reduced data to see the improvement or overheads.

Figure 3.11 shows the comparison of data reduced using MM (MM Reduced) and with the data obtained after applying ZIP compression on the reduced data (MM

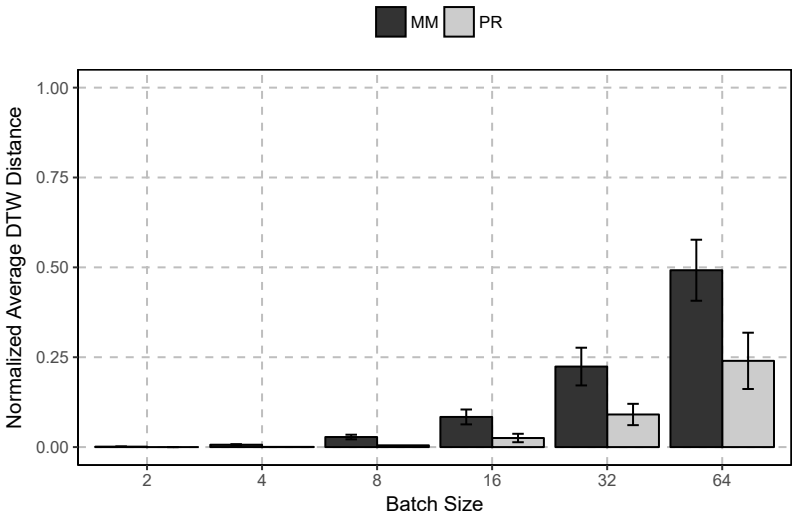


Figure 3.10: Comparison of normalized DTW distance for data reconstruction using Polynomial Regression (PR) and Markov Models (MM) methods.

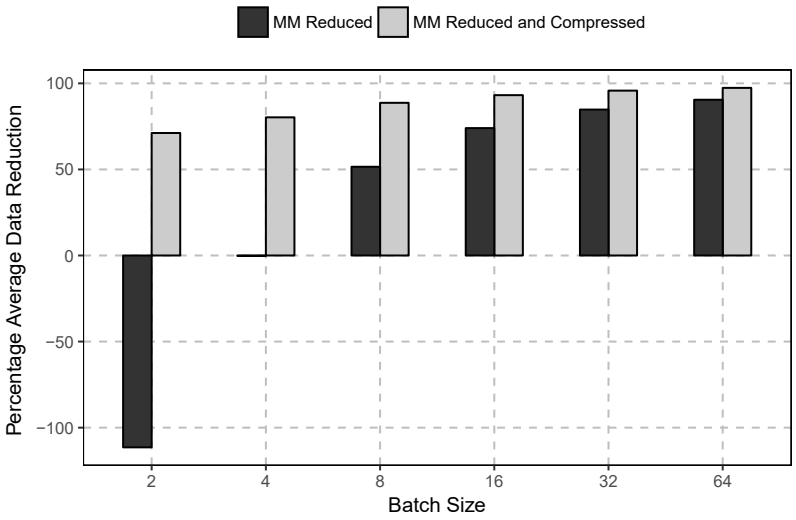


Figure 3.11: Comparison of MM Reduced data with MM Reduced and Compressed data. The MM Reduced is the data obtained by applying MM method and MM Reduced and Compressed is the data obtained after applying ZIP compression on the reduced data obtained from MM reduction method.

Reduced and Compressed) for different batch sizes. We observed the most significant data reduction due to ZIP compression in lower batch sizes particularly 2, 4, and 8. This is because, the reduced data contains the count of transition in the transition matrix which is either 0 or 1 in case of 2 states, 0, 1 or 2 in case of 4 states and 0 to 7 in case of 8 states. However, most of the values in these matrices consist of 0 due to no transition from one state to other, hence the zip compression further reduces the data. After batch size 32, we observed that the difference between MM Reduced and MM Reduced Compressed data is less than 8% on average. The maximum data reduction with zip compression is observed with 2 state MM model which is 98.24% on batch size 64. The ZIP compression helps to further reduce the data but for large batch sizes the effect of ZIP compression is not significant.

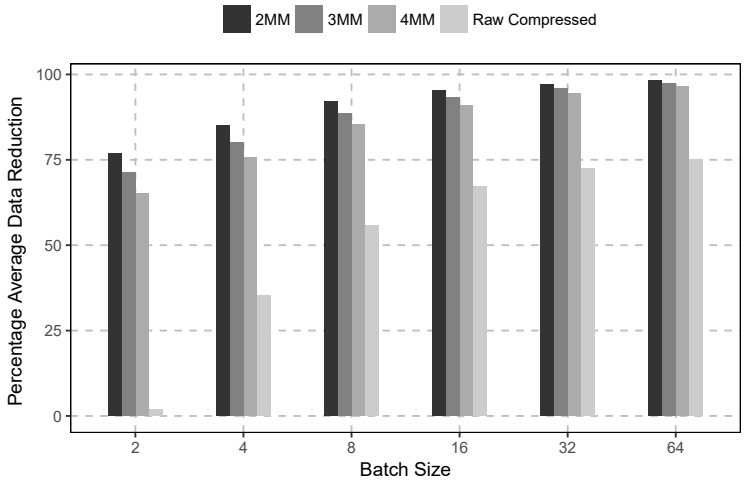


Figure 3.12: Comparison of ZIP compression on raw data and ZIP compression on data reduced using MM method.

Then, using a standard lossless ZIP compression algorithm as a baseline, Figure 3.12 shows the comparison of ZIP compression on the raw data with the reduced and compressed data (MM Reduced Compressed) using MM method on varying batch sizes. By just applying ZIP compression on the row data, we achieved 1.88% to 75.02% data reduction for batch size 2 to 64, However our proposed solution yields reduction from 76.95% to 96.48% with 2 and 4 state MM model from batch size 2 to 64 respectively.

3.3.5 Bandwidth Reduction using MM

Table 3.5 shows the bandwidth utilization reduction percentage using the proposed solution with 2, 3 and 4 state MM for telemetry data collection within the data center. We consider 4 bytes float data type to represent the value of actual data and state interval matrix. We observed that for the highest acceptable batch (batch size 64), the state transition matrix could have a maximum value of 63 as it contains the count of transition from one state to other. This type of data can be sent using one byte over the network which reduces the bandwidth utilization significantly. Moreover, higher batch sizes provide a notable reduction in bandwidth utilization within the data center. For example, batch size 32 and 64 yields significant bandwidth reduction varying from 71.88% to 93.75% for different number of MM states.

Table 3.5: Percentage bandwidth reduction within data center using MM method.

Batch Size	2MM	3MM	4MM
2	-100	-212.50	-350
4	0	-56.25	-125
8	50	21.88	-12.50
16	75	60.94	43.75
32	87.50	80.47	71.88
64	93.75	90.23	85.94

Table 3.6: Comparison of Raw and Reduced Storage (GB) for 30 days and Bandwidth (Kbps) using 2-State MM with batch size 16.

Data Center Type	Total Racks	Nodes per Rack	Total Telemetry Sources	30 days Storage (GB)			Bandwidth (Kbps)			
				Raw Uncompressed	Compressed	Reduced Compressed	Raw Uncompressed at Rack	Reduced at Rack	Raw Uncompressed at Datcenter	Reduced at Data Center
small	100	30	300,000	5,793.57	1,158.71	270.56	187.50	46.88	18,750	4,687.50
medium	250	40	1,000,000	19,311.90	3,862.38	901.87	250	62.50	62,500	15,625
large	500	60	3,000,000	57,935.71	11,587.14	2,705.60	375	93.75	187,500	46,875

3.4 Benefits of the Proposed Solution in Different Data Centers

Typical telemetry metrics consist of hardware performance counters up to a hundred events [113] related to CPU, memory, network, disk, temperature, etc. In this section, we study the effect on bandwidth utilization and storage space using the proposed solution to monitor and collect the telemetry data in the different sizes of data centers. We considered three different data centers namely small, medium, and large in which we assume that 100 different telemetry metrics are collected from each computing node after a discrete time intervals. A typical telemetry metric takes 8 Bytes to store the information including the timestamp. We selected 2 state MM model with batch size 16 as our purposed solution because we achieve 95.33% of reduction in storage, 75% of reduction in bandwidth with 92% of accuracy and Figure 3.13 shows the comparison of percentage storage space required for raw uncompressed, compressed and reduced compressed data and it also shows the percentage bandwidth utilization for raw uncompressed and reduced data for the purposed solution.

Table 3.6 shows the storage and bandwidth reduction for all three types of data centers and highlights the merits of the proposed solution. A small data center with 3000 computing nodes which are deployed on 100 racks. Where each rack hosts 30 computing nodes. To store one day of telemetry data we require 193.11 GB storage, and in a month that storage requirements increase to 5.65 TB. To store one day of telemetry data on a medium sized data center, e.g. counting with 10K computing nodes, deployed on 250 racks (considering 40 nodes per rack), we require 643.73 GB storage, and in a month the storage requirements increase to 18.85 TB. Similarly, for a large data center, e.g. with 30K computing nodes deployed on 500 racks, we will require 1931.19 GB storage space for one day, and in a month the storage requirements increase to 56.57 TB.

The proposed method of 2-state Markov Model with a batch size of 16 reduced the storage requirement to 0.26, 0.88, and 2.64 TB for small, medium and large-scale data centers respectively with a 92% reconstruction accuracy. The actual rack level bandwidth utilization for small, medium and large data centers are 187.50, 250, and 375 Kbps respectively which are significantly reduced to 46.88, 62.50, and 93.75 Kbps.

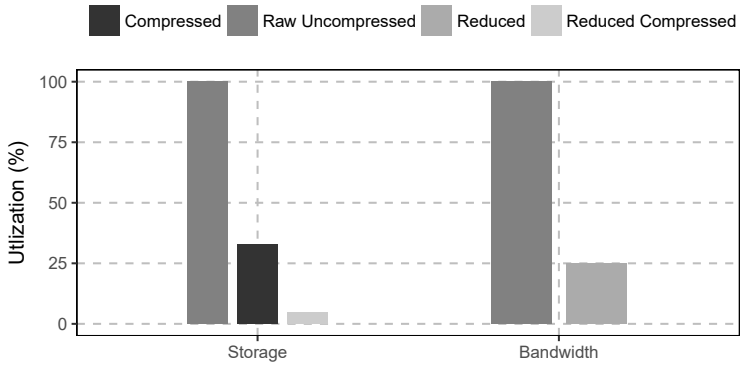


Figure 3.13: Comparison of storage usage percentage for raw uncompressed, compressed and reduced compressed (left side). Comparison of bandwidth utilization percentage for raw uncompressed and reduced data (right side).

Similarly, the actual data center level bandwidth utilization for small, medium and large sizes are 18750, 62500 and 187500 Kbps respectively. Our purposed system reduces this bandwidth utilization to 4687.50, 15625, and 46875 Kbps for small, medium and large-scale data centers respectively.

3.5 Related Work

There have been several efforts to reduce exponentially growing digital data for efficient management [59, 72, 44, 111, 121, 114, 24, 128]. Different methods were proposed including dimensionality reduction, forecasting models, and compression methods. For example, Bhuiyan et al. [9] proposed an IoT framework for event detection and data reduction at data collection time which helps to minimize data transmission across the network and also reduces energy consumption. The proposed framework detects fire events using sensors and rule-based methods. Wu et al. [125] developed a dictionary-based compression technique to split the incoming numeric data stream into fixed size blocks and compares them with the already stored blocks using Kolmogorov-Smirnov (K-S) statistical test to measure the similarity. When they identified any existing block similar to the incoming new block, they discarded the incoming block and kept a reference of the old block to be able to regenerate the data, thus helping to significantly reduce the required storage. Another work by Egri et al. [37] use dimensionality reduction of multidimensional time series data. Their approach introduces graph-based clustering using the cross-correlation between the time series data. The authors focus on identifying connections among various performance metrics in order to reduce the number of performance metrics to track.

A recent work [91] uses a correlation-based method to reduce the data center's monitoring data. The authors identify the correlation between different measurement metrics using Bayesian Network models learned from historical data and proposed to use linear regression between correlated metrics. Bayesian Network models are directed acyclic graphs (DAG) showing the relationship between metrics in the form of dependant and independent metrics. In this method, the authors reduce the sampling rate of dependent metrics and predict them using linear regression for a given duration which helps to reduce the data at the data collection stage. Another recent work by Yu

et al. [130] proposed a method to reduce data sent by edge nodes of IoT devices to the cloud for reducing data transmission time. The method modeled the incoming data as multivariant normal distribution and used Kalman filter to predict mean vector and covariance matrix of the distribution. Both the edge nodes and cloud predict the same values using identical Kalman filters. If the predicted data at both ends do not meet the confidence interval then data is uploaded from fog to cloud layer otherwise predicted values are used which helps to reduce the data movement from fog to cloud layer.

A most recent work [80] in data reduction addresses the problem of transmitting data in smart energy metering infrastructure. The authors proposed a framework to monitor data of energy consumption using smart meters and then aggregate the data for a fixed batch intervals. Then use an already learned forecasting method to compare the new data. If the new data is comparable with the forecasting model then the new data is not sent to the cloud. However, if the forecasting model and the new data are different then they send the data to the cloud and also update the forecasting method. The proposed system is adaptive according to incoming data as it does not rely on a single forecasting method rather it changes its method to suit the current data. However, this work does not address the data storage requirement optimization and only focuses on one dimensional energy consumption data.

Most of the existing works are based on either dimensionality reduction, forecasting models, or compression methods to reduce data. Our work proposes a novel Markov chain-based telemetry data reduction and reconstruction system to efficiently reduce network bandwidth utilization and storage space.The data reduction is performed in real-time without reducing the dimensions of input telemetry streams and without using forecasting models that need to update or change whenever input streams are changed. The proposed Markov Chain Models reduce the input data significantly without updating the model. Moreover, our purposed system performs more reduction compared to dictionary based ZIP compression methods. Other monitoring tools such as Ganglia [78] uses round robin database (RRD) which is a circular buffer based database. The disadvantage of these types of databases is that they drop the old data as new arrives. Moreover, an RRD file may contain multiple round robin archives(RRA) and the resolution of data in each archive is different, thus loosing the precision and details whereas, in our proposed solution, the resolution remains same for all the historical data and without dropping the old data. As Markov models are one of the simplest methods as compared to others (dimensionality reduction or forecasting methods), the overhead of converting batches into models requires very low computation as it involves only counting the transitions and does not require heavy training or updating the models which usually require in forecasting models. To the best of our knowledge, the current state of the art methods does not perform the reduction in real-time for data centers telemetry streams using Markov Chain Models. We investigate the use of Markov Chain Models and compare it with Polynomial Regression and ZIP compression with different settings to reduce the data significantly and reconstruct the data with high accuracy for data centers telemetry streams.

3.6 Final Considerations

Typically, data centers are being monitored continuously to provide better services to end users and this continuous monitoring of data centers generate a huge amount of telemetry data. The magnitude of telemetry data generated within data centers increases the storage space and bandwidth utilization requirements which is hard to

manage with the passage of time. Therefore, we require new algorithms, tools and systems to manage the dramatically growing data. This chapter introduces a novel method to reduce the bandwidth utilization and storage space requirement based on Markov Chain Models. The experimental results show that the proposed method beats the results of baseline method in which polynomial regression is used to reduce the telemetry data. The PR method takes more storage space due to the high precision of coefficients. This chapter also evaluated the effect of different batch sizes, the number of states in markov model and degree in polynomial regression. The results show that telemetry data can be significantly reduced with large batch sizes however this affects the reconstruction accuracy. Therefore an appropriate selection of batch size is necessary and we analyzed that 2-state MM model with batch size 16 can reduce the 95.33% storage space and 75% bandwidth utilization with 92% accuracy using the proposed solution. Moreover, we can obtain similar reduction for other data sets also as this technique is not dependent on the selected data set because in our proposed solution, we are only storing count of transitions with state boundaries instead of actual data. For future work, we are focusing on adaptively identifying the batch size and the number of states in MM to further reduce space and increase the reconstruction accuracy. We also plan to use one Markov Model per metric at the data center level for recurring workloads having similar resource usage requirements.

Chapter 4

Telemetry Prediction by Adaptive Prediction Models

This chapter introduces the method which dynamically selects the best prediction model for estimating and forecasting cloud resource utilization for a given recent time window of observed resource utilization based on time series features as the **second contribution** of this Ph.D. thesis. Efficient methods for estimating resource utilization in data centers can significantly ease self-management and usage optimization for both users and providers. Users can dynamically adjust the leased resources to minimize costs for hosting their applications while maintaining the desired performance and service quality [52]. Further, accurate estimates of resources utilization enable the providers to efficiently allocate virtual machines (VM) and other virtual resources to workloads, migrate VMs to consolidate or balance resource usage [126, 36], plan in advance resource capacities [112, 17], also take awareness of energy requirements in advance for expected workloads and users [90, 67]. In this chapter, we focus on classical machine learning approach. The resource utilization of data centers is a low dimensional data, and traditional machine learning methods can be effectively used for estimations. We propose a novel *adaptive model selector* method, to dynamically identify the best prediction method for estimating resource utilization of data centers, from a bag of trained methods with different characteristics and accuracy over different data center behaviors. The data center telemetry contains burstiness behavior which represents sudden spikes and peaks of resource utilization. In general, it is challenging to predict the burstiness behavior, and we address this issue with the help of an adaptive selection of an appropriate prediction method at every estimation step. After some experiments and model selection, we chose Random Decision Forests (RDF) as the best mechanism for learning the expected accuracy for each candidate predictor.

Our proposed method trains on the statistical features of historical resource utilization and predictor correctness for sliding windows of a specific size, to identify which predictor will produce the best forecast given the current resource utilization. We evaluate our method by comparing its decision and forecasting capabilities with baseline methods, using data sets from Alibaba and Bitbrains monitored data centers. Results show that the proposed method outperforms the baseline methods for both of the data sets. Notice that in this work we focus on CPU resource consumption as the primary resource on high-performance computing data centers, but our solution can be used to predict utilization of all system resources. Moreover, we evaluated the proposed solution with data centers however our solution is generic enough and it can be applied to other types of workloads with minimal changes.

To test and validate our selective multi-method approach, we have conducted experiments using the Alibaba [3] and Bitbrains [11] data center utilization data sets, comparing them to baseline methods, also against single method approaches.

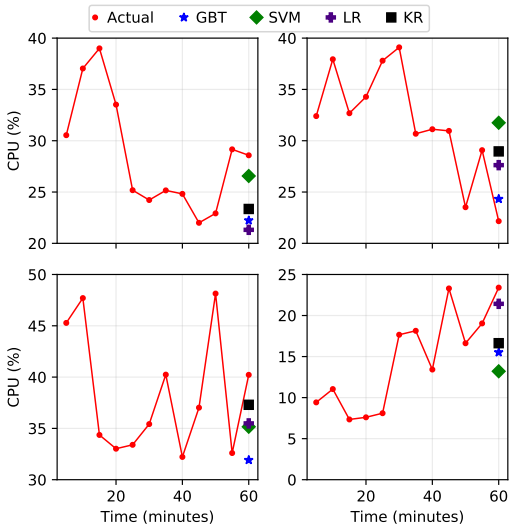


Figure 4.1: CPU estimation using different methods and scenarios for Alibaba data set. Different predictors yield better estimation, each for different scenarios.

The baseline methods used are well-known machine learning methods used by current state-of-the-art approaches, like Linear Regression (LR), Support Vector Machines (SVM), Gradient Boosting Tree (GBT) and Gaussian Process also known as *Krigin* (KR). Figure 5.2 shows the motivation to adaptively select an appropriate method to estimate resource utilization for different scenarios effectively. The figure shows CPU utilization estimations using different methods for Alibaba data set of four different machines. Each estimation method is trained using 55 minutes time interval data and estimated the utilization for the next 5 minutes. We observed that different predictors yield better estimation, each for different scenarios, on forecasting resource consumption. Therefore, building a system able to identify the best predictor for forecasting resource utilization at each time segment becomes attractive.

The main contributions of this chapter are as follow:

- A novel method to dynamically select the best prediction model for estimating and forecasting cloud resource utilization for a given recent time window of observed resource utilization.
- A comparison of different baseline models, currently used in the state-of-the-art, as candidate models for resource utilization estimation, aside of validation for the presented approach.
- Analyze the impact of different window sizes on the proposed resource estimation systems.

4.1 Proposed System Overview

The overall proposed system is illustrated in Figure 4.2. Different steps are numbered and labeled to explain the working flow of the system. The system work in the following steps:

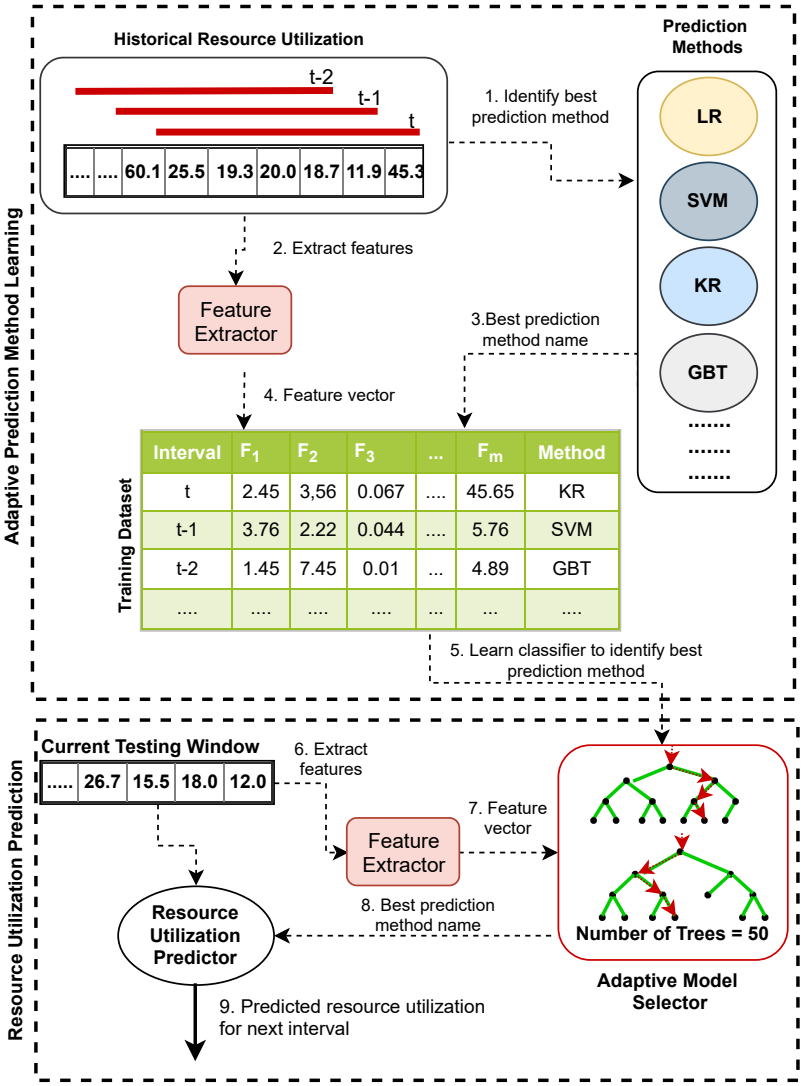


FIGURE 4.2: Purposed system overview to learn adaptive model selector and using it to estimate the data center resource utilization.

- Historical resource utilization logs of the data center are divided into sliding windows of a fixed size consists of the last k intervals. Then each sliding window data is used to fit different prediction models including Linear Regression (LR), Support Vector Machine (SVM), Kriging (KR), and Gradient Boosting Tree (GBT) to predict the next interval resource utilization. The system selects the prediction method yields a minimum prediction error for the given sliding window data.

- For each sliding window, the system identifies a specific set of features as explained in Section 4.3.

- The selected features and identified prediction methods are logged as training data. For each historical sliding window, the training data set contains the corresponding feature vector and the best prediction method.

- Once training data is prepared, the system builds a classifier using Random Decision Forest (RDF) to predict the best model for a given sliding window data. We call this classifier "Adaptive Model Selector". We explain this in Section 4.2.2.

- Once the Adaptive Model Selector is trained than the system predicts the data center resource utilization in real time. For the current time interval t, system select last k observation to extract features and then use the Adaptive Model Selector to identify the best prediction method to predict the resource utilization for the $t + 1$ time interval.

- The selected prediction method is used to train a regression model using the last k interval's observed resource usage data to estimate the resource utilization for the $t + 1$ future time interval.

4.2 Machine Learning Methods

In this work we use Machine Learning (ML) techniques for two main purposes: first, predict future workload behaviors and traces; second, from a set of ML methods and a context, choose one that predicts the workload better. The work presented here focuses on different algorithms for regression used to predict the workload, while a trained decision maker selects at each time a regression model that is expected to produce the most accurate prediction.

4.2.1 Workload Prediction Methods

In order to predict future workload features, we explore a diversity of Machine Learning techniques commonly used in the literature, ones more complex than others with different properties each . The learned regression models are to predict our target variable which is next data point in time series from known input features [118] such as skewness, standard deviation, kurtosis, autocorrelation for different lags, absolute sum of changes, and others. As the data we are dealing is in the form of a time series, evaluation of prediction must be based not just on accuracy but also on the significance of results which is often a difficult problem on regression analysis.

Our presented methodology shows a multi-model approach, where different models are trained, each one with a different set of strong and weak properties. The models are applied to a dynamic window to predict future interval workloads. The studied models for workload prediction are: Linear Regression, Support Vector Machines

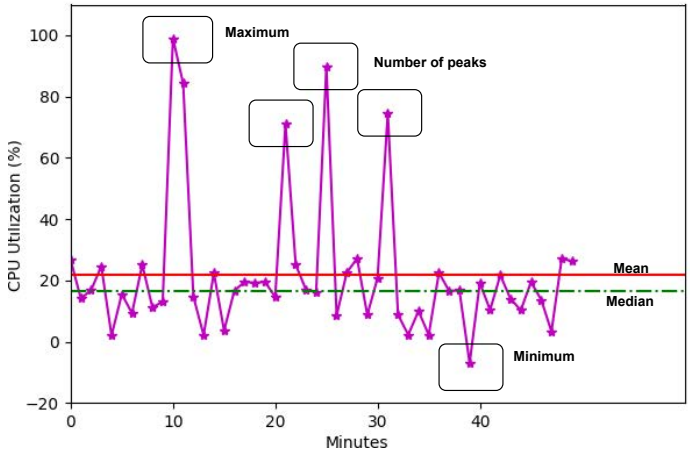


Figure 4.3: Example of time series features that are extracted from TSFRESH [23] library. These features consist of statistical and time series features such as minimum, maximum, variance, standard deviation, number of peaks, autocorrelation at different lag intervals, entropy, kurtosis, skewness, fourier transformation, mexican hat wavelet transformation, and etc.

for regression, Gradient Boosting, and Gaussian Process Regression and discussed in detail in section 2.4.

4.2.2 Adaptive Model Selector (AMS)

On multi-model methodologies, different regression models produce predictions altogether, and a trained expert system decides which prediction is followed, or how they are aggregated into a final prediction. Such a trained expert can be a machine learning model, like in Boosting methods. In our proposed solution, before producing workload predictions, we use a trained decision maker to choose the best predictor to be used. The decision maker will classify each scenario into the best-expected predictor for it.

Our decision maker input will be features [118] such as skewness, standard deviation, kurtosis, autocorrelation for different lags, the absolute sum of changes, etc., and it will output the regression method which is expected to be the best. At each time step, the decision maker predicts the best regression model and then produces the workload prediction using the predicted regression model. In section 2.4, we present the different classification models studied in this work.

4.3 Feature Extraction and Selection

Appropriate features can play an important role to improve the prediction accuracy of machine learning models. In our data set, the resource utilization of data centers is available as a time series data. We explore multiple ways to extract time series features from the given data set which includes manual extraction, automatically extraction by the help of open source libraries such as Cesium [18], TSFRESH [23]. However we selected TSFRESH as it provides us most useful and a comprehensive set of time series features which is not available in any other library. Time Series Feature extraction based on scalable hypothesis tests (TSFRESH) [23, 45] is an open-source Python library available to extract features for a given time series data. In

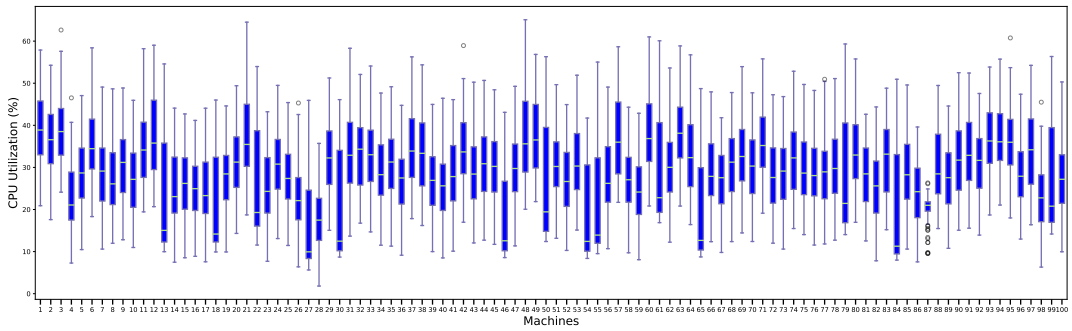


Figure 4.4: Box plot of CPU utilization for randomly selected 100 machines from Alibaba data set.

our proposed system, we used TSFRESH to extract features for data center resource utilization data available as time series. TSFRESH automatically calculates a large number of time series characteristics based on scalable hypothesis tests.

Figure 4.3 shows some of the features that TSFRESH extracts for the given time series data. It provides hundreds of statistical and time series features including minimum, maximum, variance, mean, standard deviation, sum of values, autocorrelation of the specified lags, measure of non linearity in the time series, Mexican hat wavelet, first and last location of minimum and maximum, number of peaks, quantile, and sample entropy etc. However all of these features are not necessary, and appropriate features should be identified to improve the performance of machine learning methods [49, 20].

The proposed system filters the features obtained from TSFRESH using another open-source library available for feature selection [40]. We selected this library because it includes a comprehensive set of functions to filter the features by using different approaches to identify the most appropriate features for time series classification. The library provides five different methods to filter features for `missing values`, `single unique values`, `collinear features`, `zero importance features`, and `low importance features`. However, in our proposed system, we only used three methods to filter the features obtained through TSFRESH. First, we apply three methods to filter the feature. First, we apply `single unique value` method which remove the features with identical unique values. Second, we apply `identify collinear` which remove the features which are highly correlated with one another. We used 98% correlated threshold in this method to ensure only remove the features which correlated 98%. Finally, we apply `zero importance features` which uses Gradient Boosting Machine (GBM) learning model to identify the features which have zero importance for the given set of features. After applying these methods, we obtain one hundred and six features in total which include standard deviation, kurtosis, skewness, absolute some of the changes, auto-correlation at different lags, partial auto-correlation at different lags, the first location of minimum, linear least-squares regression[71], and many others.

4.4 Experimental Evaluation

In this section, we explain the data sets used to evaluate our proposed method, the details about the experiments used to validate it, the baseline methods used for comparison, and the used evaluation metrics.

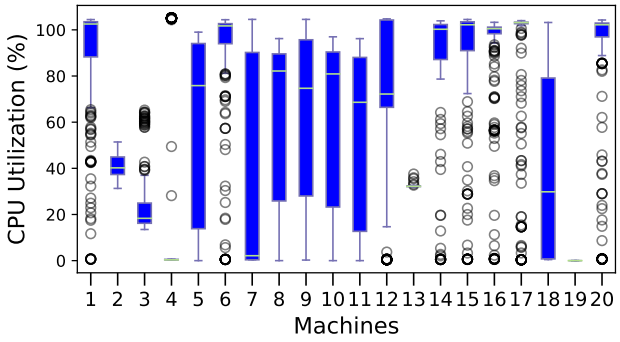


Figure 4.5: Box plot for Bitbrain data set of 20 randomly selected VMs for one-day data.

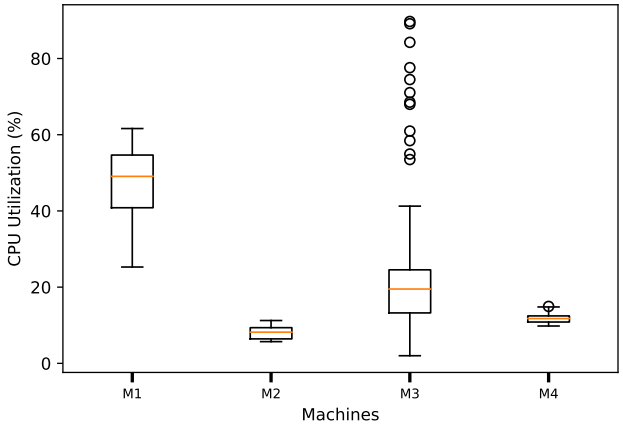


Figure 4.6: Box plot for CPU utilization of selected four machines with different characteristics from the Alibaba data set. M1=high load, M2=low load, M3=high variation, and M4=low variation.

4.4.1 Data sets

Alibaba data set

The first data set we use is the *Alibaba cluster logs* [3], publicly available, containing performance traces of 1,313 machines for 12 hours duration. The Alibaba monitored cluster provides interactive services and batch processing workloads. The metrics represented are CPU, memory and disk utilization for all machines, aggregated on 5-minute averages. For simplification purposes, we are focusing on and experimenting with CPU time series. The average CPU utilization in the Alibaba data set is 26.46%, with a standard deviation of 10.66% CPU. Figure 4.4 shows a CPU utilization sample for 100 randomly selected machines from the data set.

BitBrains data set

The second data set we use is the *Bitbrains data set* [11], publicly accessible, containing performance logs of 1,750 VMs for 30 days of data. The Bitbrain monitored cluster provides interactive services and batch processing workloads. The metrics represented are CPU, memory, network and disk utilization for all the virtual machines, aggregated on 5-minute averages. From this data set, we randomly selected 20 VMs with average CPU utilization greater than 30%, as most of the VMs with low usage do not show critical metric patterns or utilization tends to be constant on the lowest part of the spectrum demand. Figure 4.5 shows the box-plot for one-day data of the average CPU utilization for the selected machines.

Google data set

The Google cluster traces [48] are the publicly available traces published by Google. To create the CPU and the Memory utilization, the tasks of each job were aggregated by summing their CPU and Memory consumption every five minutes in a period of 24 hours. The dataset was extracted over the first ten days period by filtering the utilization of CPU and memory from 5 to 90 percent, resulting in a total of 1,600 VMs [83]. We randomly selected 500 VMs from this data set for the experiments and the average CPU utilization in the selected data set is 21.89%, with a standard deviation of 3.63% CPU.

4.4.2 Methodology

For the current experiments, we are using the Alibaba data set to show a comprehensive evaluation of the proposed solution. Whereas, the BitBrains data set is used for testing to show that the proposed methodology does not over-fit to the primary data set (Alibaba).

To train and validate the machine learning models in the AMS classifier, we are using a random split of 80% data for training, and the remaining 20% for validating the models. The model is trained offline with 80% of data. The test data also includes the four machines which are discussed in the following paragraph.

As applications running on data-centers can have different profiles, we selected four machines from the Alibaba data set with very distinct CPU demands, to test the resource estimation on different demand behaviors. Figure 4.6 shows the box-plot of CPU utilization of the four selected machines: Machine M1 serves a workload demanding high CPU resources; machine M2 serves a workload requiring low CPU

resources; machine M3 serves a workload requiring CPU resources with highly fluctuating demand; and finally machine M4 serves a workload requiring CPU resources with low fluctuating demand.

Our proposed solution is then compared with the aforementioned baseline methods, proposed by Liu et al. [73], using Linear Regression (LR) and Support Vector Machines (SVM) methods to estimate adaptively CPU utilization of VMs. The combination of the two methods, LR for the slow-changing workloads and SVM for the fast-changing ones, are here labeled as "Liu" method. In addition, we also add the methods namely LR, SVM, Kriging (KR) and Gradient Boosting Tree (GBT) to consider for comparison with ours.

4.4.3 Experimental Details

Adaptive Model Selector Evaluation

The Adaptive Model Selector (AMS) is in charge to estimate which of the available ML algorithms will provide better modeling for the current data being monitored. We performed a set of experiments to evaluate different methods to make such estimation by comparing different classifiers namely Random Decision Forest (RDF), Gradient Boosting Tree (GBT), Multi-layer Perceptron (MLP), K-Nearest Neighbors(k-NN), Gaussian Naive Bayes (NB), and Support Vector Machine (SVM) with linear kernel. These classifiers are trained and validated using the Alibaba data set. We trained all classifiers on 80% of the entire Alibaba data set and then tested on the remaining 20% data to compare and identify the best classifier to used in AMS.

Training and validation data will be structured in time windows, as explained in Section 4.4.3. The classifiers are evaluated through True/False Positive Rates (TPR and FPR), accuracy, recall, f-measure, and precision. We also consider the performance of the AMS by measuring the training time, prediction time, and the size of the model on disk.

Resource Estimation Evaluation

Finally, when integrating the different techniques of model selection and resource modeling, we perform a set of experiments to evaluate the resource estimation using our proposed adaptive ensemble. The final goal is to identify, on-line, the best regression method that will build a prediction model for estimating the resource utilization of the next future interval, given the current monitored data.

As this problem is a regression one, we evaluate the complete mechanism using the Root-Mean Square Error (RMSE) as shown on Equation 4.1 to show how our method deviates from the truth, also the Mean Absolute Error (MAE) as shown on Equation 4.2 to show the absolute magnitude of the produced error. Here a_t is the true CPU utilization and p_t is the estimated CPU utilization at time interval t, and n is the number of performed estimations.

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} (a_t - p_t)^2}{n}} \tag{4.1}$$

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^{n} |a_t - p_t| \tag{4.2}$$

Table 4.1: AMS evaluation results using different classifiers for Alibaba data set.

Classifier	TPR	FPR	TNR	FNR	Precision	Recall	F-measure	Accuracy
KNN	0.62	0.11	0.88	0.37	0.65	0.65	0.65	0.65
NB	0.33	0.22	0.77	0.66	0.38	0.31	0.29	0.31
RDF	**0.65**	**0.10**	**0.89**	**0.34**	**0.68**	**0.68**	**0.68**	**0.68**
GBT	0.48	0.16	0.83	0.51	0.55	0.53	0.51	0.53

Table 4.2: Time and space efficiency of AMS using different classifiers for Alibaba data set.

Classifier	Training Time (sec)	Prediction Time (sec)	Prediction Time per Request (ms)	Size (KB)
KNN	3.23	593.61	17.017	255283.2
Naive Bayes (Guassian)	0.59	0.13	0.004	7.5
RDF	**57.43**	**0.51**	**0.015**	**201523.2**
GBT	186.45	0.28	0.008	140.9

Again, training and validation data will be structured in time windows (Section 4.4.3), and then for each sliding window, we use the AMS to identify the best regression method to estimate the resources for the following intervals.

Window Size Sensitivity

A specific observation window size is required to train the AMS. In this experiment, we evaluate the effect of different window sizes on the proposed solution by quantifying the estimation error using Alibaba dataset. We tested window sizes of 20, 40, 60, 80 and 90 minutes of data to train and validate the proposed solution for resource estimation. To segment the data set into training/validation sets, we performed a random split 80%/20%. We organized the training data into windows of the aforementioned sizes, and evaluate the models and ensemble, using the RMSE and MAE metrics to quantify the effect of each different window size.

4.5 Experimental Results

4.5.1 AMS Evaluation

The Adaptive Model Selection method is evaluated through the aforementioned quality metrics for different classifiers, and check not only accuracy but also the performance requirements for each, like time for training and predicting, and size of the resulting model. The hyperparameters used for each classifier are tuned using grid search.

Table 4.1 shows the evaluation results of the AMS using the selected features of the raw data set to identify the best prediction method for CPU resource utilization estimation using Alibaba data set. The AMS is evaluated by true positive rate (TPR), false positive rate (FPR), true negative rate (TNR), false negative rate (FNR), precision, recall, f-measure, and accuracy. TPR is the ratio of right predictor correctly classified as right. FPR is the ratio of right predictor incorrectly classified as the wrong predictor. TNR is the ratio of wrong predictor identified as wrong predictor and FNR is defined as the proportion of wrong predictor incorrectly identified as right predictor. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The high value of precision indicates the low false

Table 4.3: RMSE and MAE for resource estimation using the purposed system for Alibaba data set.

Method	RMSE	MAE
GBT	4.57	3.43
LR	5.12	3.87
SVM	5.63	4.23
Kringing	5.26	3.99
Liu [73]	5.34	3.94
Proposed	**3.32**	**2.29**

positive rate. Recall, also known as sensitivity is the ratio of correctly predicted positive observations to all the observations in the actual class. The high value of recall indicates the low false negative rate. F-measure considers both precision and recall and it is the weighted average of precision and recall. The high value of F-measure represents balances between precision and recall. The low value represents the improvement in one measure at the expense of others. Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observations to the total observations.

Table 4.1 table shows TPR, FPR, TNR, FNR, precision, recall, f-measure, and accuracy for using kNN, Naive Bayes, RDF, and GBT as classification methods in AMS to identify the prediction method which can be used to estimate the CPU resources with high accuracy. The RDF outperforms all other classifiers. We observed that KNN, as second best classification method in AMS also provides comparable and closest results to RDF.

To profile the time and space efficiency of different classifiers for AMS using Alibaba data set, we profile training time, testing time, and the size of the trained model on the disk. Table 4.2 shows the time and space efficiency of AMS using different classification methods. We observed Naive Bayes classifier is efficient by consuming the least time to train and test the AMS. Whereas, the classification performance of Naive Bayes is significantly lower than RDF specifically for precision, recall, f-measure, and accuracy.

Although kNN classification performance is comparable to RDF, however, training, testing, and disk size of AMS using kNN is worst comparing to other classification methods. The RDF training and test time are reasonably good, and it outperforms other classification methods for all evaluation metrics. Therefore, we chose RDF classifier to use in our proposed AMS.

Figure 4.7 shows the Receiver Operator Characteristics (ROC) curve using RDF with AMS for different classes. ROC curves for all the classes are better than the random classifier. We observed that the proposed AMS with RDF efficiently classifies the test data for all the classes. The area under the ROC curves is 0.84, 0.89, 0.90, and 0.90 for SVM, LR, GBT, and KR labels respectively.

Overall, we observed that using RDF in AMS performs excellently to identify appropriate prediction method to use adaptively for the given data for resource estimations.

4.5.2 Resource Utilization Estimation

Table 4.3 shows RMSE and MAE for CPU utilization estimations on test data of Alibaba data set for the proposed and baseline methods. The proposed method outperforms all baseline methods by yielding minimum RMSE and MAE.

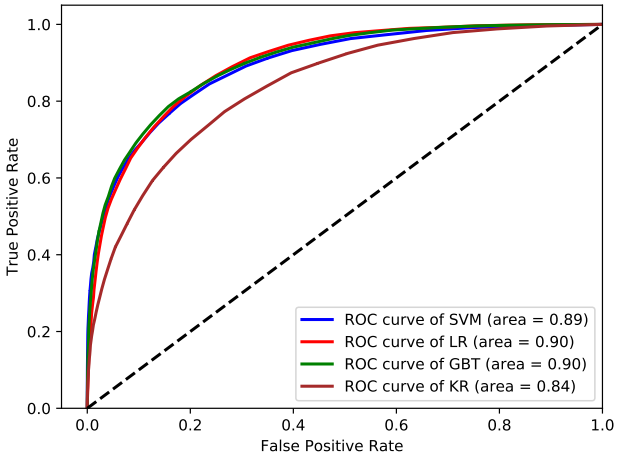


Figure 4.7: ROC curves using RDF with AMS for different classes.

To compare the proposed method with baseline methods we normalized the RMSE with relative to the proposed solution, as shown in Figure 4.8. We observed 27%, 35%, 37%, 38%, and 41% less estimation error comparing to GBT, LR, KR, Liu, and SVM baseline methods.

Figure 4.9 shows the box plot for absolute error computed for each estimated CPU utilization using Alibaba data set for the proposed and baseline methods. We observed the proposed method outperforms the baseline methods to minimize the absolute error.

Figure 4.11 shows the recommendations proposed by AMS as a function of time for the selected four machines. The proposed method dynamically select the most appropriate prediction model based on time series features of the recent window. Figure 4.10 shows the comparison of actual and estimated CPU resources using the proposed system for the four selected machines. The proposed method to estimate the CPU utilization shows significantly closer to the actual resource utilization for all of the machines serving a significantly different type of workloads.

To quantify and visualize the error for each estimation, we show absolute error frequency computed for machines M1 and M3 using baseline and proposed methods in Figure 4.12 and Figure 4.13 respectively. Where M1 serves a workload demanding high CPU resources consistently, and M3 served a workload requiring CPU resources with fluctuating demand. We observed that the proposed method always yield minimum error to estimate the CPU resource utilization for a different type of workloads. We also observed that the proposed method yield minimum absolute error for each estimation comparing to the baseline methods for both M1 and M3 machines.

4.5.3 Window Size Sensitivity Analysis

Figure 4.14 shows the RMSE and MAE for different windows sizes with the proposed system to estimate CPU resource estimation. We observe that increasing window size reduce the estimation error till window size 60, however, after that the error starts rising. The 20 minutes window size only contains four observations to fit the prediction models for estimation which yields a maximum error. This experiment identifies that 60 minutes window size is optimal to use with the proposed system to minimize the

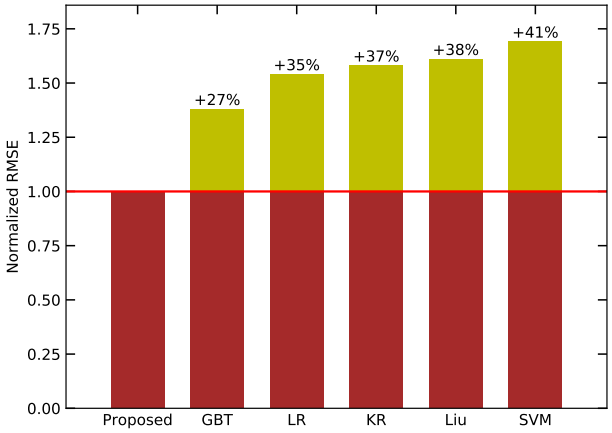


Figure 4.8: Comparison of normalized RMSE for baseline methods with the proposed method using Alibaba data set.

Table 4.4: RMSE and MAE for resource estimation using the purposed system for Bitbrains data set.

Method	RMSE	MAE
GBT	9.74	2.85
LR	15.01	6.03
SVM	19.94	7.19
Kringing	15.80	6.05
Liu [73]	19.80	7.09
Proposed	**9.13**	**2.57**

estimation error. Therefore, in all of our experiments, we used 60 minutes window size with the proposed and baseline methods.

4.5.4 Evaluation Using BitBrains Data set

Table 4.4 shows RMSE and MAE for estimating CPU utilization on test data using Bitbrains data set for the proposed and baseline methods. Instead of repeating all the experiments to identify the best classifier, we validate the proposed solution using the same configuration which we found best for Alibaba data set. We observe that the proposed method outperforms all baseline methods by yielding minimum RMSE and MAE.

To show the comparison of the proposed method with baseline methods, we normalized the RMSE with relative to the proposed solution. Figure 4.15 shows the comparison of baseline methods with the proposed solution by calculating normalized RMSE for the Bitbrains data set. We observed 6%, 39%, 42%, 54%, and 54% less estimation error comparing to GBT, LR, KR, SVM, and Liu baseline methods respectively.

Figure 4.16 shows the box plot for absolute error computed for each estimated CPU utilization using Bitbrains data set for baseline and proposed methods. We observed that the proposed method produces less absolute error compared to the baseline methods.

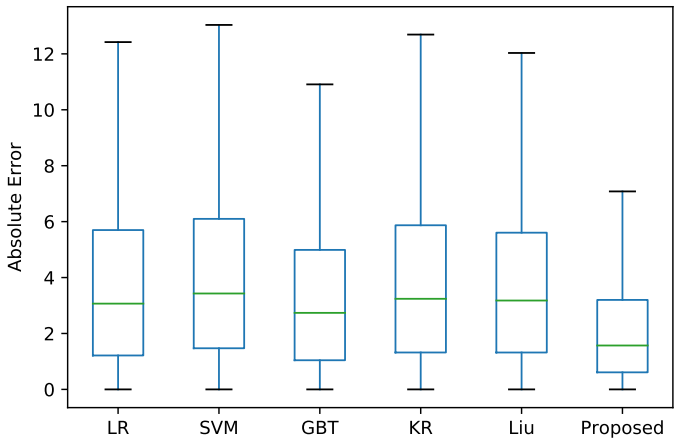


Figure 4.9: Box plot of absolute error computed for each estimation using baseline and proposed methods for Alibaba data set.

Table 4.5: MSE and MAE for resource estimation using the purposed system for Google Cluster data set

Method	RMSE	MAE
GBT	2.31	1.24
LR	2.40	1.32
SVM	2.35	1.28
Krining	2.28	1.24
Liu	2.26	1.24
proposed	**2.22**	**1.14**

4.5.5 Evaluation Using Google Data set

After performing additional experiments using the Google dataset, the same used by Liu [30], we realized that while such dataset presents a behavior with less variance than Bitbrains (more than 80% of the machines report standard deviations below 4 in a range of 0 to 100), and all methods behave with similar good accuracy, also both methods Liu's and ours are better than the individual machine learning algorithms. But then, for the Bitbrains and Alibaba datasets with higher variance and more extreme behavior, and while Liu's method does not adapt that well, our method still does and improve the individual algorithms. Table 4.5 shows RMSE and MAE for estimating CPU utilization on test data using Google data set for the proposed and baseline methods. The proposed method outperforms all baseline methods by yielding minimum RMSE and MAE.

4.6 Related Work

Data center resource utilization and workload prediction is an active research area. Recently, there have been several attempts to use machine learning methods for predictions of data center resources. For example, recent work by Kim et al. [58] proposed an ensemble approach which uses multiple predictors together to produce an output. The proposed ensemble technique uses Linear Regression, SVM, ARMA, and ARIMA together to predict future workload for the data centers by dynamically determining

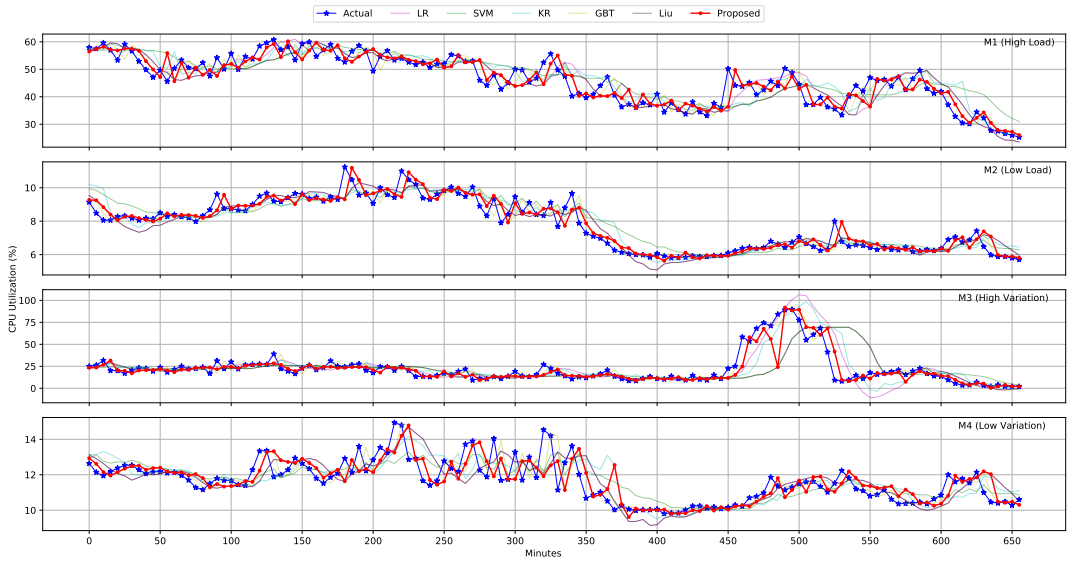


Figure 4.10: Actual vs proposed method CPU prediction for Alibaba data set for four selected machines. M1 = Heavy workload, M2 = Low workload, M3 = High variation, M4= Low variation. The window size used to train the prediction model is 60 minutes.

the weight of each predictor using the regression method. Another recent work by Rahmanian et al. [97] also proposed an ensemble-based approach to predict CPU utilization of application usage of VMs. The proposed approach uses automata theory to adjust the weight for each predictor in the ensemble method to predict the CPU usage. Subirats et al. [110] proposed an ensemble-based prediction strategy which forecasts the infrastructure energy requirement by predicting the future CPU utilization of VMs. Their ensemble-based approach uses the moving average, exponential smoothing, linear regression, and double exponential smoothing methods. Chen et al. [21] propose an ensemble model based on the fuzzy neural network to predict the resource demand. They use the second moving average (SMA), exponential moving method (EMA), autoregression model (ARM), and trend seasonality model (TSM) as base predictors. Cetinski et al. [19] combine statistical and machine learning methods to predict application specific workload volume. Tseng et al. [117] used a multi-objective genetic algorithm to forecast resource utilization and energy consumption in data centers. Jiang et al. [54] proposed ensemble prediction mechanism to predict the cloud workloads for capacity planning in data centers. They used five prediction algorithms named as moving average, autoregression, artificial neural network, support vector machine, and gene expression programming to predict the future workload estimations.

There have been several efforts to use typical time series solutions to predict data center resource utilization. For example, Rodrigo et al. [15] used autoregressive integrated moving average (ARIMA) method to predict the arrival rate for the applications hosted on the cloud. Liao et al. [70] use typical time series prediction methods namely autoregressive moving-average, moving average, and auto-regressive together as an ensemble approach to predict CPU usage of VMs. The proposed method combines the output of time series prediction techniques as input to another linear prediction model to predict CPU utilization of VMs. Vazquez et al. [119] used various time series prediction models to forecast the number of requests which helps in the

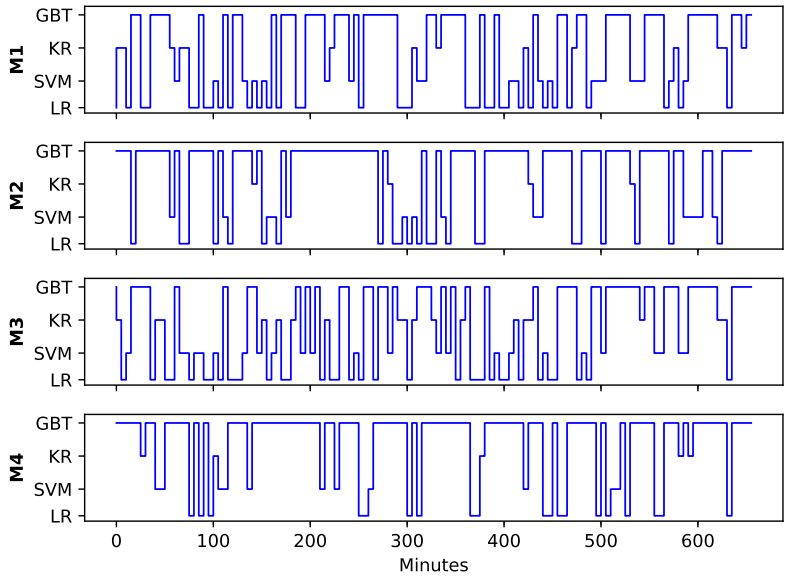


Figure 4.11: Model selection of Adaptive Model Selector (AMS) for Alibaba data set for four selected machines. M1 = Heavy workload, M2 = Low workload, M3 = High variation, M4= Low variation.

dynamic scaling of cloud resources proactively. For this purpose, they evaluated the autoregressive model (AR), moving average model (MA), simple exponential smoothing, double exponential smoothing, automated ARIMA method, and neural network autoregression method. Dmytro et al. [29] use ARIMA to forecast load on the cluster which helps in scheduling the data center resources by migrating the VMs. Fang et al. [39] used ARIMA to predict the future CPU utilization and a number of requests for the applications hosted in the cloud.

There have been several efforts to employ deep learning methods for predicting data center resource utilization. For example, Zhang et al. [135] use autoencoders to predict the CPU utilization of VMs. The authors used tensor rank decomposition technique to reduce the training time by compressing the input parameters. Feng Qiu [96] used a deep belief network using multiple-layered restricted Boltzmann machines (RBMs) and a regression layer to predict the CPU usage of VMs. The RBMs are used to extract the high-level features, and the regression layer is used to predict CPU utilization. Zhang et al. [136] also use RBMs to predict CPU and RAM utilization in data centers. They use backpropagation as global supervised learning to minimize the loss function. Mason [76] predict the CPU consumption of the host by using evolutionary Neural Networks (NN). To train the network weights of neural networks, they used Particle Swarm Optimization (PSO), Differential Evolution (DE), and Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). Song et al. [108] use long short-term memory (LSTM) model to predict the host load. To train the recurrent networks, the authors used truncated back-propagation through time technique. Duggan et al. [35] predict host CPU utilization by using Recurrent Neural networks. They also use the back-propagation through time (BPTT) technique to train the network.

The work in this area most relevant to ours [73] adaptively picks either Regression (LR) or Support Vector Machine (SVM) predictors to estimate CPU utilization of VMs. The proposed method dynamically select LR for slow changing workloads and

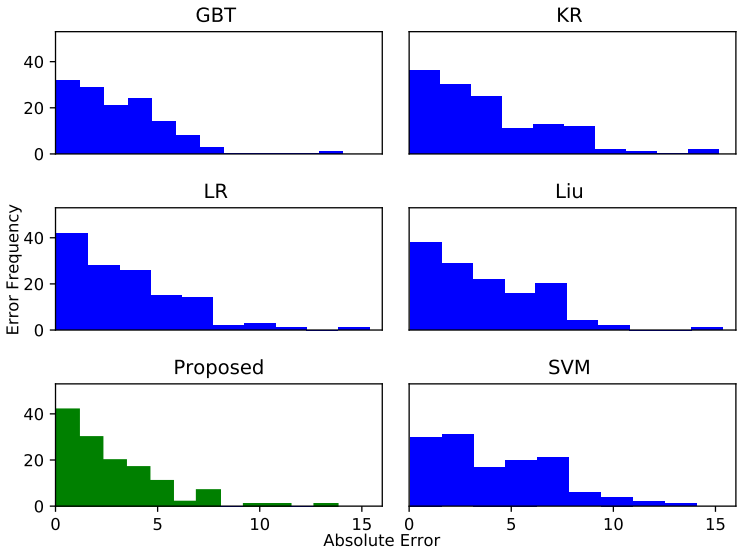


Figure 4.12: Absolute error frequency of CPU utilization estimation for machine M1 (High Load).

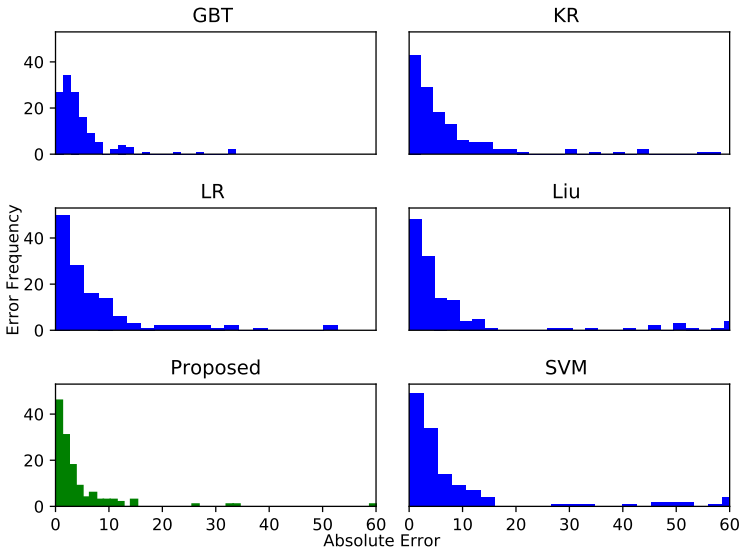


Figure 4.13: Absolute error frequency of CPU utilization estimation for machine M3 (High Variation).

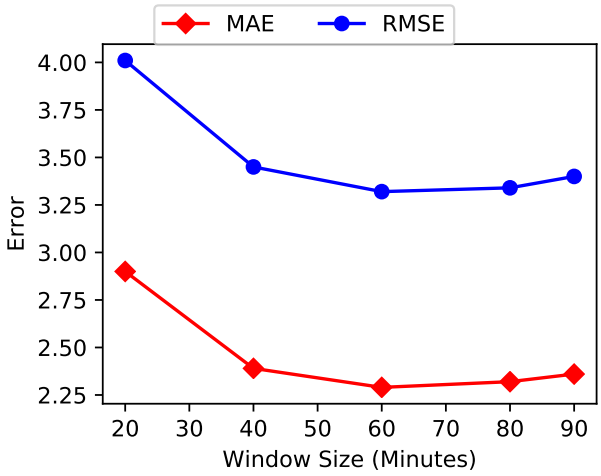


Figure 4.14: RMSE and MAE using different window sizes with the proposed system for resource utilization estimation.

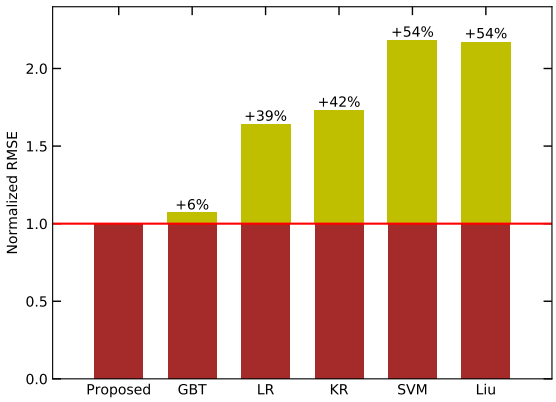


Figure 4.15: Comparison of normalized RMSE for baseline methods with the proposed method using Bitbrains data set.

SVM for rapidly changing workloads.

Moreover, most of the existing works use ensemble-based approaches in which multiple estimation methods are collectively used to produce the final output. In this approach, all the models need to train every time as the final output is dependent on all the models whereas in our proposed solution the final output is produced using only a single machine learning predictor and does not require training of n models all the time. Our approach uses four different estimators and dynamically identifies the estimator using a machine learning approach and time series features. To the best of our knowledge, no existing work which uses time series features to adaptively identify and use the best prediction method to minimize the estimated error of cloud resource utilization.

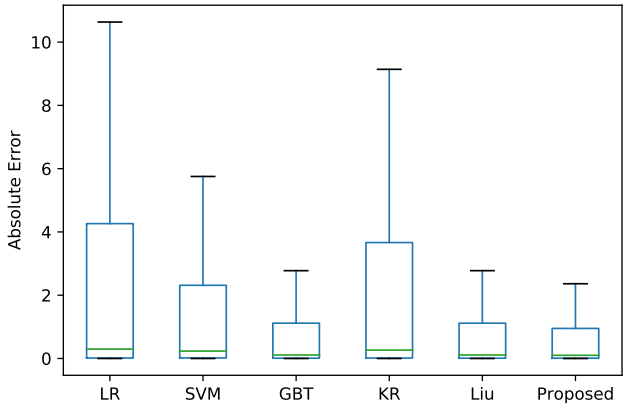


Figure 4.16: Box plot of absolute error computed for CPU utilization estimation using baseline and proposed methods for Bitbrain data set.

4.7 Final Considerations

Accurate prediction of data centers resource utilization is challenging due to extremely diverse nature of workloads running on top of heterogeneous infrastructure in data centers Therefore building new methods for estimating resource utilization in data centers is an active research problem. To tackle this problem, this chapter introduces a novel method of future resource estimation by selecting the most appropriate machine learning method adaptively based on statistical series features of recent observation windows. The proposed method uses random forest classifier to select the most appropriate prediction model to be used for predicting the resources on the next time interval. The statistical features are extracted from the historical observations to train the RDF along with the best method for each time window. We observe that the proposed method outperforms the existing state of the art method and improve the prediction accuracy from 6% to 27% when evaluated on real traces collected from Alibaba and Bitbrains data-center monitoring data sets. The chapter also includes the effect of window sizes when modeling and predicting the resources. To effectively estimate the future resource utilization, we observed that, 60 minutes of historical resource utilization observation could be used to build the prediction model. The novelty of this method lies in identifying the appropriate machine learning methods for each specific scenario over time and future work will focus on investigating adaptive window size for modeling and predicting data center resource utilization.

Chapter 5

Adaptive Window Size Selector for Prediction Models

This chapter presents the method to dynamically identify the best sliding window size to train the regression model for estimating and forecasting cloud resource utilization as the **third contribution** of this PhD thesis. In this chapter, we addressed this challenging problem and proposed a method using deep learning approach to adaptively select the best observation window efficiently to minimize the estimation error. The proposed adaptive sliding window identification method limits the observation window which can be used by any estimation method to train the model for predicting next interval observations. We have evaluated the proposed adaptive sliding window identification with the commonly used machine learning namely Linear Regression, Ridge Regression, Least Absolute Shrinkage and Selection Operator (LASSO), Elastic Net, and Non-negative Least Square, and Support Vector Regression for multiple publicly available data sets. The proposed adaptive sliding windows outperform the fixed sliding windows using all estimation methods by significant margins for all data sets.

To show the motivation for using an appropriate observation window size for resource estimation, we conducted a preliminary experiment using Alibaba cluster traces [3], randomly selecting a virtual machine (VM) and using its CPU utilization to study the effect of different observation window sizes using different estimation methods. Figure 5.1 shows the CPU utilization estimation for 31st-time interval (from a 30-time observation series) using Linear Regression (LR), Polynomial Regression (PR), Support Vector Regression (SVR), and Elastic Net (EN) for observation window sizes 5, 10, 15, and 20. We observed that window size 10 and 15 yield best estimations using LR, window size 20 gives the minimum estimation error using SVR, RR gives best estimation result for window size 10, and EN yields minimum estimation size for window size 15. These results show that the appropriate observation window size can play an important role to minimize the estimation accuracy for different estimation method. We advocate that any estimation algorithm can improve the estimations using appropriate observation window size. The problem to automatically identify the appropriate window size at every estimation step to train the model is challenging.

To further elaborate on the advantage of using adaptive sliding windows for resource estimation, we study the effect of estimation performance using different fixed sliding window sizes and adaptive observation windows. We used LR as an estimation method and used the entire CPU utilization data of the selected VM as a time series to evaluate different fixed observation windows sizes. We identified the optimal observation window size using exhaustive search among window sizes 2 to 30 and selected the optimal window size which yields maximum accuracy as optimal window size at

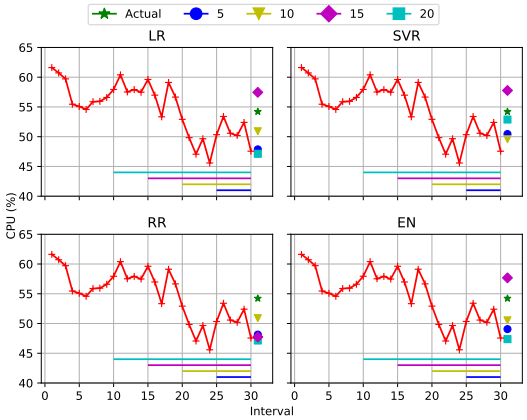
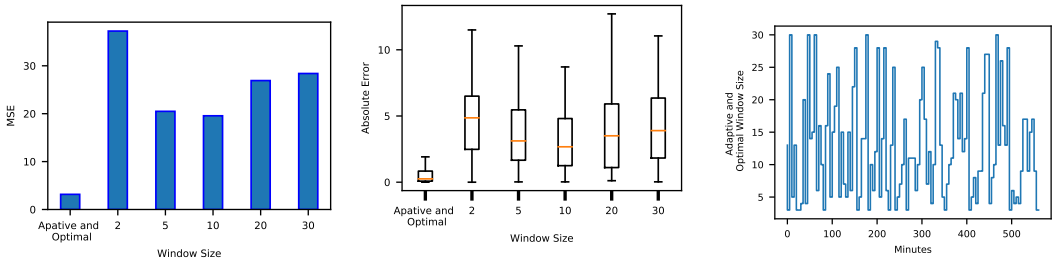


Figure 5.1: CPU estimation using different size of sliding windows for various machine learning predictors.



(A) Comparison of MSE **(B) Box plot for absolute errors** **(C) Optimal window sizes**

Figure 5.2: Evaluation of fixed sliding windows of different sizes and adaptive optimal sliding windows for estimating data center CPU resources.

every estimation step. Figure 5.2a shows the mean squared error (MSE) for static sliding windows of sizes 2, 5, 10, 20, 30, and optimal window size using adaptively in each estimation step. Figure 5.2b shows the box plot for absolute errors obtained for each estimation step for different fixed sliding window sizes and optimal window sizes identified adaptively. Figure 5.2c shows adaptive the sliding window sizes identified adaptively at each estimation step. We observed that the adaptive observation windows yield significantly minimum estimation error comparing to static fixed size sliding windows. However, it is challenging to determine the appropriate observation window size efficiently at each estimation step for maximizing prediction accuracy.

The main contributions of this chapter are as follows:

- Propose a novel method to dynamically identify the best sliding window size to train the prediction model for estimating data center resource utilization.
- Evaluation and comparison of the proposed method with different baseline methods, currently used in the state-of-the-art, as candidate methods for window size estimation, aside of validation for the presented approach.
- Extensive experimental evaluation using three publicly available data sets of different real data centers.

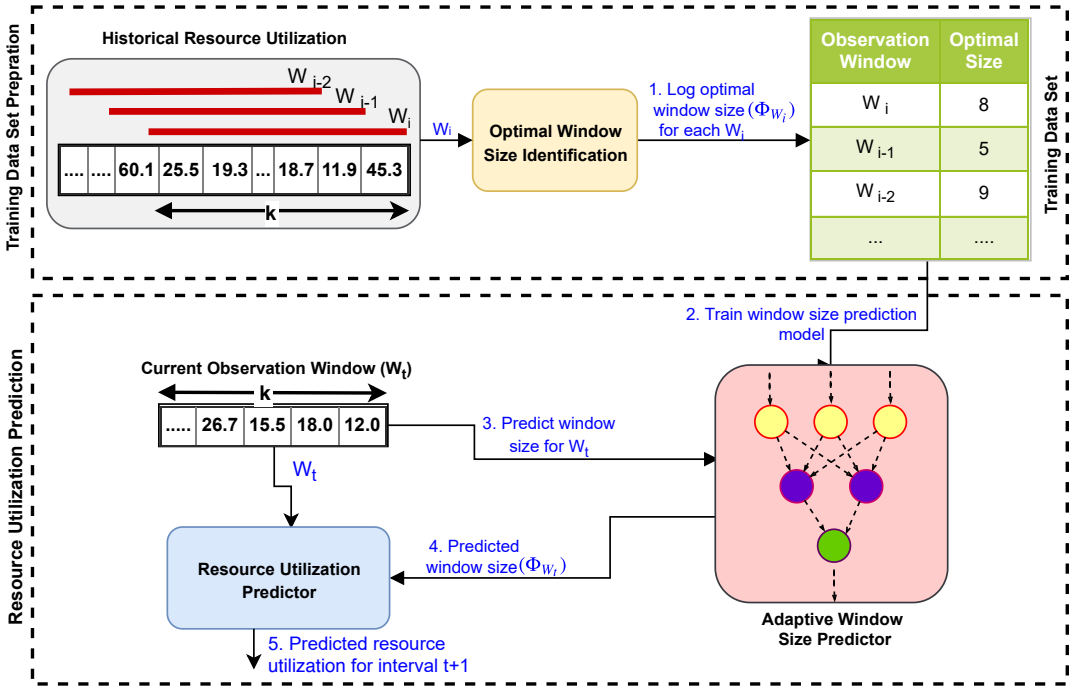


Figure 5.3: Purposed system overview to learn adaptive window size predictor and using it to estimate the data center resource utilization

5.1 Proposed System Overview

The proposed system architecture is illustrated in Figure 5.3. Different steps are numbered and labeled to explain the flow of the system. The proposed system works in the following steps.

i. Historical resource utilization logs of the data center are divided into sliding windows of a k fixed size intervals. Each sliding window at time interval t is called an observation window. Our objective is to identify an appropriate size for the observation window to train a resource prediction model with minimal prediction error.

ii. For each sliding window W_i, the system identifies the optimal window size to predict the next interval's resource consumption with minimal prediction error. In this phase, the next interval's consumption data for each sliding window is known. The system performs a linear search to identify the window size with minimal prediction error. The system checks the observation window into $k-1$ sub windows, start from length 2 to k. Each sub-window is used to estimate the resource consumption, and the size of the sub-window with minimum prediction error is identified as optimal window size Φ_{W_i} for the corresponding sliding window W_i, where $2 \geq \Phi_{W_i} \leq k$.

iii. Each sliding window W_i and the identified corresponding optimal window size Φ_{W_i} is recorded as a training data set to predict the window size for resource utilization estimation.

iv. Once training data is prepared, the system train a deep neural network to predict the best window size for a given sliding window data. We named this component as "*Adaptive Window Size Predictor*".

v. Once *Adaptive Window Size Predictor* is trained then the system uses it to identify the best window size for each time interval t from the current observation window.

vi. The predicted window size is used to select the number of observations from the k data points of the current observation window for training a regression model to predict the resource utilization for the next time interval $t + 1$.

In summary, the proposed system consists of "Adaptive Window Size Predictor" which predicts the number of most recent observations (window size) require to build estimation model for better estimation accuracy. This predictor is trained offline using historical resource utilization of all servers of a data center without machine identifications. The predicted window size from 'Adaptive Window Size Predictor' will be used as input to another machine learning algorithm to build resource estimation model for predicting next interval resource usage.

5.2 Adaptive Window Size Predictor Using Deep Learning

We use deep learning method to adaptively select the window size for machine learning models. Deep Learning is a set of machine learning methods based on neural networks, where those networks have more than one layer of transformations between the input data and the output data to be matched. MultiLayer Perceptrons (MLP), the proposed neural networks, consists on the transformation of an input data-set $X_{N,I}$ (a matrix of N observations and I features) to an output data-set $\hat{Y}_{N,J}$ (a matrix of J transformed features per observation). A *Perceptron* (the usual neurons on MLPs) is an artifact that processes the input as $\hat{Y}_{N,I} = G(F(X_{N,I}))$, where $F(X_{N_I}) = X_{N,I} \cdot W_I + B$, and W_I is a matrix of weights to be adjusted, B is a vector of biases, and $G(X)$ is a function that in regression can be the Identity or in classification can be a *sigmoid* function. A "single hidden layer" MLP consists on an array of Perceptrons (the hidden layer) processing that input as $X'_{N,H} = G(F(X_{N,I}))$, where $F(X_{N,I}) = X_{N,I} \cdot W_{I,H} + B_H$, being H is the number of perceptrons on the hidden layer. The purpose of a layer is to find a non-linear relation between its inputs and outputs, then the results can be aggregated in the output layer as $\hat{Y}_{N,J} = G(F(X'_{N,H}))$. A deep MLP concatenates different hidden layers, and the output of a layer is the input of the next one. In that case, each layer i processes $\hat{X}^i_{N,H^i} = G(F(X^{i-1}_{N,H^{i-1}}))$, where H^i is the number of neurons in that layer. Training a MLP consists in finding the function MLP that transforms $MLP(X) = \hat{Y} \sim Y$, where Y is the real output data-set to be matched. Weight matrices W and bias vectors B are adjusted using Gradient Descent, by iteratively passing data through the network forth and back. The goal of adding multiple layers to a network is to learn latent patterns that a simple non-linear function can not represent, through combinations of multiple non-linear relations.

While other approaches attempt shallow architectures, here we propose the use of deep neural networks, mainly because of the stochastic and non-linear nature of the workload data. As seen in other works referring to resource utilization and Cloud and virtualization management, like [135, 76, 124], the use of deep neural networks on the prediction of resource usage and management has proven to be reasonably

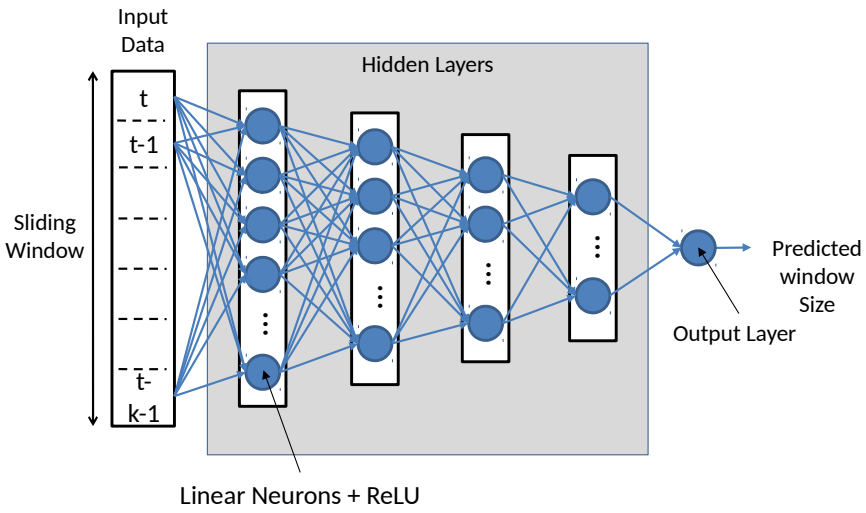


Figure 5.4: Schema for a 4-Hidden Layer Deep Neural Network on our Time-Series

cost-efficient and accurate, and this brings us to attempt these techniques for our adaptive window prediction scenario. After testing different architectures and hyper-parameters, including the number of layers and number of neurons per layer, using a grid-search method, we selected the model resulting in the most accurate without ending with an overkill model. Such a model has an input of N elements as the size of the sliding window, and an output of 1 value, resulting on the prediction. The deep network has 4 hidden layers, with $\langle 23, 15, 10, 5 \rangle$ hidden units each layer, with *rectified linear unit* (ReLU) activation function and normally random initialization. It is optimized using the *Adam* method for stochastic gradient descent [60], using the *Mean Squared Error* (MSE) as quality metric. Figure 5.4 shows the basic schema of our neural network. Figure 5.5 shows the effect of the number of epochs on the mean squared error for training and validation data sets. The validation data set consist of 20% of the training data set. The network has been trained for 500 epochs, with a batch size of 250 units, out of $11,8563$ total elements on the training data-set.

The input data-set, being data a time series, is generated by sliding a window of size k, generating an observation for each window movement. The window is slide 1 time-step at a time. Each observation is introduced in the network as a vector of K values, considering that the window sample is always ordered. For validation purposes, the final data-set is randomly split 80/20 for training vs. testing subsets.

5.3 Estimation Methods for Resource Utilization Prediction

Our proposed technique to identify observation windows to build estimation method works with all machine learning and statistical estimations methods. We explore machine learning methods easy to train requiring less computing power for building resource estimation model using the predicted observation windows. We explain the estimation methods in section 2.4.

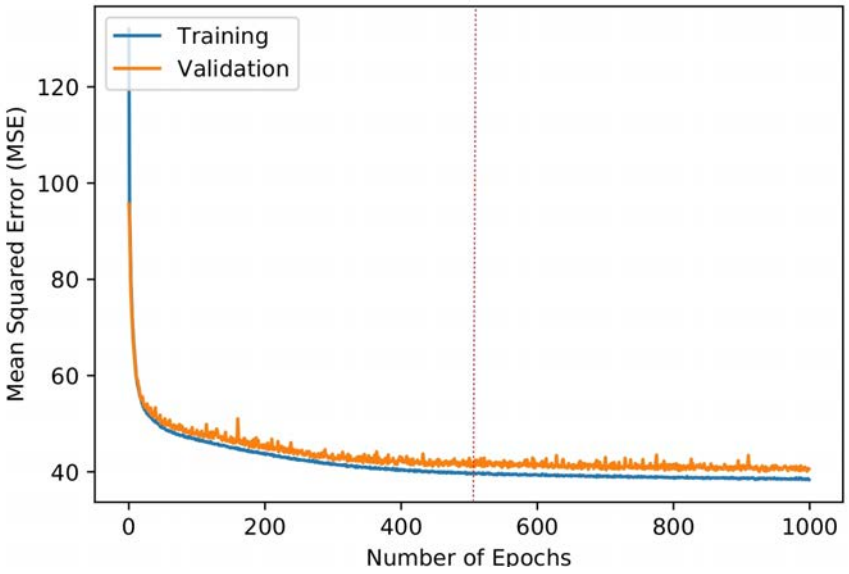


Figure 5.5: Effect of number of epochs on the MSE for training and validation.

Table 5.1: Data sets with CPU resource utilization statistics and utilization categories.

Data Set	Total Machines	Average Load (%)	Average Variation in Load	Utilization Category
Materna	520	4.44	9.29	Low
Alibaba	1,313	26.46	10.66	Moderate
Bitbrains	131	44.21	44.84	High

5.4 Experimental Setup and Design

5.4.1 Data Sets

To evaluate the proposed solution, we used three publicly available data sets representing diversified CPU resource utilization characteristics. Table 5.1 shows the total number of machines, average CPU load in percentage, average CPU load variations in the data sets. Each data set is categorized either low, moderate, or high CPU serving workloads. Materna data set is labeled as low CPU workload as the CPU load and variation is considerably low comparing to other data sets. Alibaba data set is considered as a traces of the data center serving moderate CPU workload. Whereas, Bitbrains data set is recognized as a data center serving high CPU workload. The data sets are briefly discussed in the following subsections.

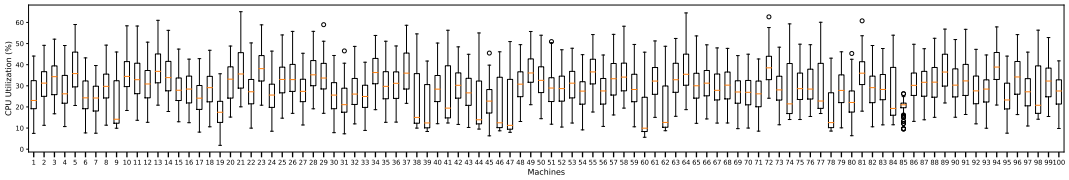


Figure 5.6: CPU utilization for randomly selected 100 machines from Alibaba data set for twelve-hour data

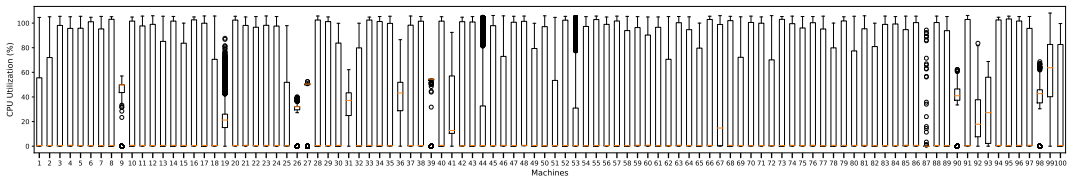


Figure 5.7: CPU utilization for randomly selected 100 machines from Bitbrains data set for one-month data

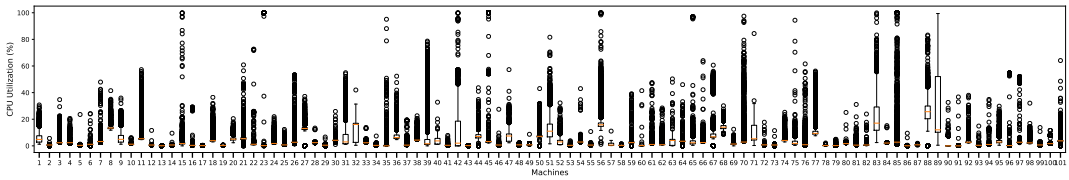


Figure 5.8: CPU utilization for randomly selected 100 machines from Matenra data set for one-month data

Materna

Materna is a service provider offering cloud services to the aviation industry. Their data center performance traces for 30 days are publicly available [79]. The data set contains resource utilization metrics for CPU, memory, network, and disk for 520 different VMs running on the data center. Each of these resource utilization metrics is sampled for 5 minutes average utilization. Figure 5.8 shows CPU utilization for 100 randomly selected machines from the data set. Most of the machines CPU utilization remains below 5%; therefore, we considered this data set representing low CPU workload serving data center.

Alibaba

Alibaba cluster logs [3] are publicly available data set containing performance traces of 1,313 machines for 12 hours. The Alibaba cluster serves interactive and batch processing workloads. The available metrics in the data set are CPU, memory, and disk utilization for all machines representing average utilization for 5 minute time intervals. Figure. 5.6 shows a CPU utilization sample for 100 randomly selected machines from the data set which shows that most of the machines CPU utilization remains from 20% to 50%. Therefore, we considered this data set representing moderate CPU workload serving data center.

Bitbrains

Bitbrains is a service provider offer cloud services for enterprises. Their data center performance traces representing 1,750 VMs for 30 days is publicly available [11]. This data set also contains the average CPU, memory, network, and disk utilization for all the VMs sampled for 5-minute interval. From this data set, we selected VMs with average CPU utilization greater than 30% (131 VMs) to build a data set representing data center serving high CPU workload. Figure 5.7 shows a CPU utilization sample for 100 randomly selected machines from the data set which shows that most of the machines CPU utilization remains from 0% to 100%.

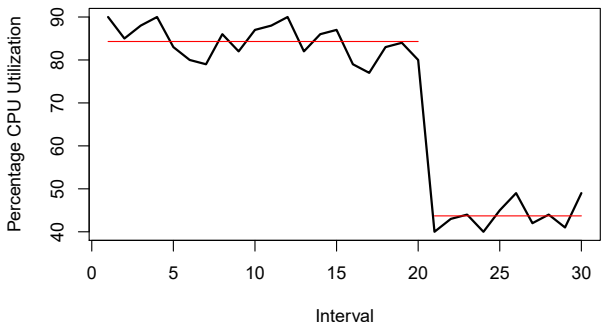


FIGURE 5.9: Change Point Detection (CPD) method to identify observation window for building resource estimation model.

5.4.2 Evaluation Criteria

To evaluation the proposed method, we used Mean Square Error (MSE) to quantify the error in resource estimation prediction. The MSE is calculated using true and estimated values using:

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(a_t - p_t)^2, \tag{5.1}$$

where a_t is the true CPU resource utilization and p_t is the estimated CPU utilization at t-th time interval and n is the total number of estimations.

5.4.3 Experimental Details

We performed three different experiments to evaluate and compare the proposed system. In **Experiment 1 (FixW)**, we evaluate the effect of different fixed size observation windows on resource estimation accuracy for all three data sets. We use 5, 10, 20, and 30 fixed observation window sizes to train and estimate the resources using machine learning methods, explained in 5.3 for all three data sets.

In **Experiment 2 (CPD)**, we employ Change Point Detection (CPD) method [101] to adaptively identify the appropriate window size for training resource estimation models. This method allows selecting adaptive observations windows to build estimation models by using only data points after the recent change point. For example, Figure 5.9 shows the use of CPD method to identify the adaptive observation window to use for training the estimation method. Assuming, current time interval is 30 and CPD provides a change point at 21st time interval for the given data then we will use observations from 21st interval to 30th interval for training an estimation model for future resource utilization estimation. The CPD method can identify multiple change points for the given time series data, however, we limit the observation window to the recent change point. In this experiment, we limit the maximum observation window size to 30 intervals which represents last 150 minutes of resource utilization observations.

In **Experiment 3 (Proposed)**, we use the proposed Adaptive Window Size Predictor (AWSP), explained in Section 5.2, to adaptively identify the best observation window to build resource estimation method. For every time interval, we provide

Table 5.2: Experiment 1 (FixW) results showing MSE for different estimation methods and fixed window sizes.

Data Set	Window Size	LR	SVR	EN	LASSO	RR	NNLS
Alibaba	30	34.28	42.15	34.06	34.02	34.28	45.25
	20	33.18	39.04	32.43	32.23	33.16	42.14
	10	**25.45**	29.5	23.92	24.09	**25.24**	30.91
	5	28	**23.72**	**22**	**23.28**	25.71	**28.49**
Materna	30	12.81	17.83	12.70	12.68	12.80	14.12
	20	11.63	15.69	11.45	11.45	11.62	12.85
	10	**11.39**	12.57	10.75	**10.95**	**11.29**	**11.98**
	5	13.02	**11.12**	**10.41**	11.81	11.91	12.70
Bitbrain	30	380.75	696.47	378.41	379.67	380.63	442.22
	20	301.83	577.35	297.75	300.46	301.47	360.44
	10	220.90	359.98	209.52	218.43	218.57	249.85
	5	**178.14**	**249.14**	**146.67**	**173.27**	**162.01**	**190.12**

the last 30 interval observations to the pre-trained AWSP and obtain an observation window size and then build the estimation model for predicting the future resource utilization estimation. In this experiment, we also limit the maximum observation window size to 30 minutes which represent the last 150 minutes of the resource utilization observations.

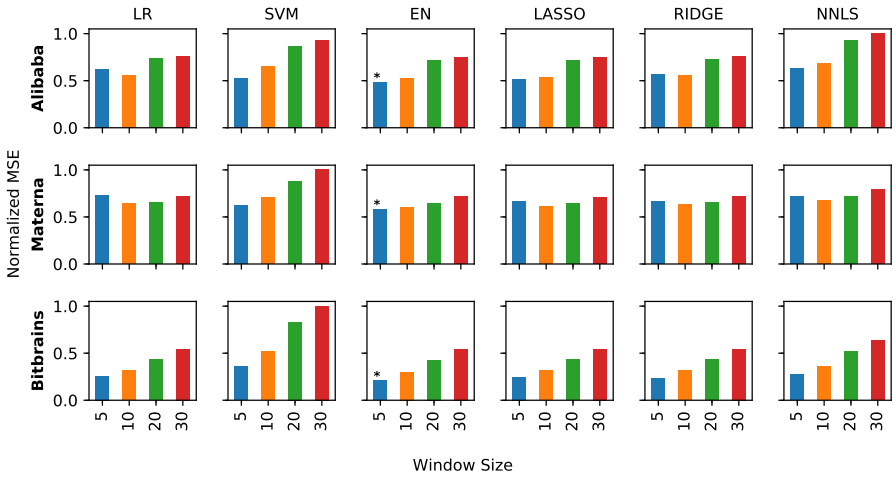


Figure 5.10: Experiment 1 results showing Normalized MSE for comparison of different estimation methods and window sizes for all three data sets.

5.5 Experiment Results

5.5.1 Experiment 1: Fixed Observation Window (FixW)

Table 5.2 shows MSE for all three test sets using different resource estimation methods and window sizes 5, 10, 20, and 30. MSE represents resource estimation error and small values of MSE show better estimation results. We observed that small window sizes like 5 and 10 yield minimum MSE. For Alibaba data set, LR and RR yield

Table 5.3: Experiment 2 (CPD) results showing MSE for different estimation methods.

Data Set	LR	SVR	EN	LASSO	RR	NNLS
Alibaba	30.40	**23.24**	25.84	26.92	27.48	28.50
Materna	20.71	**11.02**	12.35	17.65	13.36	20.44
Bitbrains	272.05	292.55	**222.73**	262.29	234.68	276.49

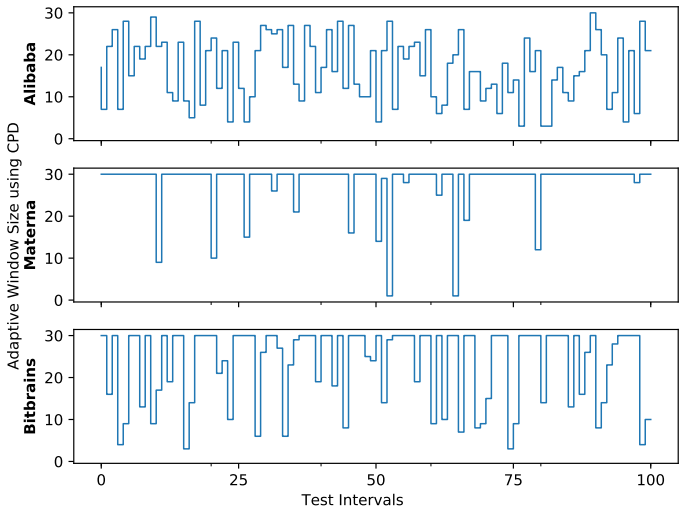


Figure 5.11: Adaptive window sizes obtained using the CPD method in Experiment 2 for first 100 test intervals of all three data sets.

minimum MSE using window size 10 whereas SVR, EN, LASSO, and NNLS produce minimum estimation error using window size 5. For Materna data set, LR, LASSO, RR, and NNLS give minimum MSE to estimate the CPU resource utilization whereas SVR and EN using window size 10 give minimum resource estimation error. For Bitbrain data set, window size 5 yield minimum error for all estimation methods. Overall EN with window size 5 outperforms all other estimation methods for all three data sets.

To compare different estimation methods and window sizes for all three data sets, we compute normalized MSE. Figure 5.10 shows the normalized MSE for the results obtained in Experiment 1. EN for window size 5 gives minimum estimation error for all data sets, whereas SVM yields worst results in all of the data sets for using window sizes 20 and 30. In general, the small window sizes with linear models show better results and exhibit that the resource utilization of data centers is locally linear. However, identifying the appropriate fixed window size to minimize the resource estimation error is a tedious and challenging task.

5.5.2 Experiment 2: Adaptive Windows Size using Change Point Detection Method

Table 5.3 shows the MSE for all three test data sets and different estimation methods using observation windows selected through the Change Point Detection (CPD) method [101]. The CPD selects adaptive observation windows to build estimation

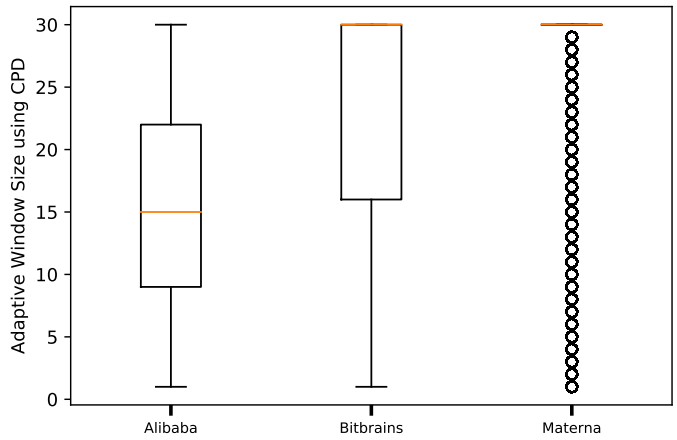


Figure 5.12: Box plot of adaptive observation window sizes using the CPD method for test data sets.

models by using data points after the recent change point from the given historical observations. SVR yields the minimum MSE for Alibaba and Materna data sets whereas EN produces the minimum MSE for Bitbrains data set using the CPD method.

Figure 5.11 shows the adaptive window sizes obtained through the CPD method for building estimation models for first 100 test intervals of Alibaba, Materna, and Bitbrains data sets. At each test interval, a maximum of 30 previous observations are passed to the CPD and dynamically limit the observation windows to train the estimation models. For Materna data set, the CPD method does not variate the observation window so frequently whereas for Alibaba and Bitbrains the observation windows are frequently variates. This shows that for low variation data set like Materna the CPD methods also gives the observation windows with low variations. To further study the adaptive observation windows identified by the CPD for all three data sets, we draw the box plot.

Figure 5.12 shows the box plot of the adaptive window sizes for the entire test sets of all three data sets. For Alibaba data set, the observation windows variate between 9 and 22 with 15 mean window size. For Bitbrains data set, in average the observation window size remains 30 and variate between 15 to 30 sizes. Whereas, Materna which represents low utilization workload with little variations in resource utilization pattern mostly use 30 observation window size and show few outliers varying from 3 to 29. This clearly shows the limitation of the CPD method for the cases where resource utilizations do not show any notable changes in the usage pattern, and the CPD favors a bigger size observation windows.

The CPD method eliminates the need to search for the appropriate observation window size for building estimation method. However, the results obtained using the CPD methods are not outperforming FixW method results.

5.5.3 Experiment 3: Adaptive Windows Using Proposed Method

Table 5.4 shows the MSE for all three test data sets and different estimation methods using the observation windows obtained through the proposed *Adaptive Window Size Predictor* method. For each data sets, all estimation methods yield comparable resource estimation results. However, EN gives the minimum MSE for Alibaba and

Table 5.4: Experiment 3 (Proposed) results showing MSE for different estimation methods using the Proposed method to identify the observation window sizes.

Data Set	LR	SVM	EN	LASSO	RR	NNLS
Alibaba	16.86	15.82	**14.97**	16.13	15.59	17.04
Bitbrains	154.70	169.21	**139.69**	156.38	148.08	156.08
Materna	10.19	10.11	9.45	9.63	9.93	**8.73**

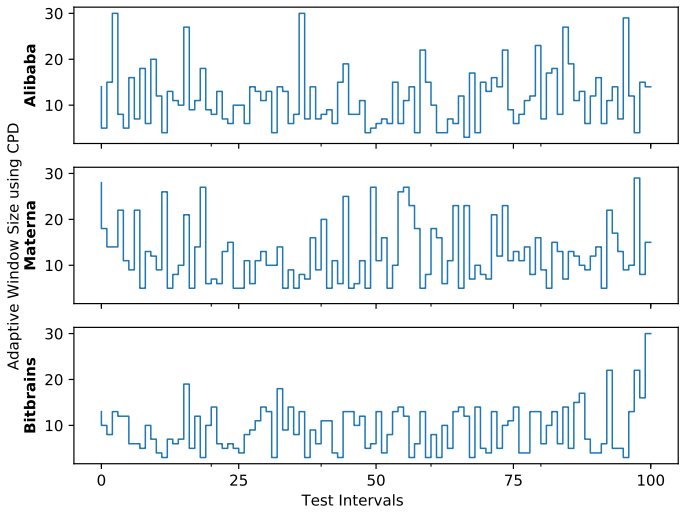


Figure 5.13: Adaptive observation window sizes obtained using the Proposed method in Experiment 3 for first 100 test intervals of all three data sets.

Bitbrains data sets whereas NNLS produces the minimum MSE for Materna data set using the proposed method.

Figure 5.13 shows the adaptive window sizes obtained using the Proposed method for training estimation models for first 100 test intervals of three representative test data sets. For all data sets, the proposed method identifies different observation window sizes even for Materna data set which represents the workload with fewer variations. The proposed method variates the adaptive observation window sizes and yields better estimation results comparing to the CPD and FixW methods.

Figure 5.14 shows box plot for the adaptive window sizes identified by the proposed method for all three test data sets. For Alibaba data set, the observation window sizes variate between 2 to 30 with 13.26 mean. For Bitbrains data set, the observation window sizes variate between 2 to 30 with 9.65 mean and Materna data set shows the observation window sizes between 5 and 30 with 12.77 mean.

The Proposed method outperforms the FixW and CPD observation window selection methods and also helps to eliminate the daunting task for searching appropriate fixed window sizes.

5.5.4 Comparison of FixW, CPD, and Proposed Method

In this section, we compare the results obtained in Experiment 1 (FixW), Experiment 2 (CPD), and Experiment 3 (Proposed). Table 5.5 shows the normalized MSE for all three experiments and data sets. For each data sets, the maximum MSE among

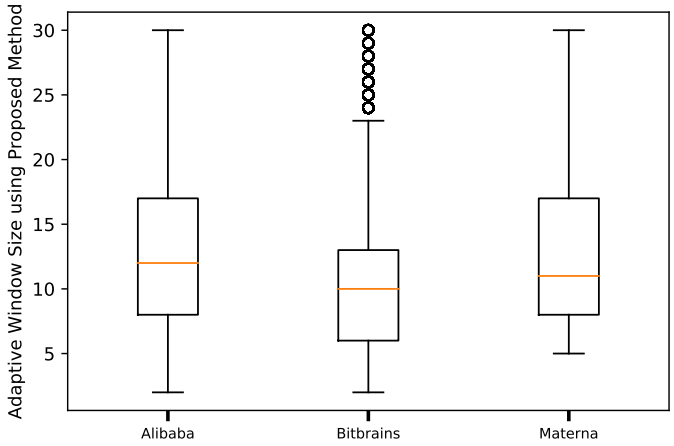


Figure 5.14: Box plot of adaptive observation window sizes using the Proposed method for test data sets.

Table 5.5: Normalized MSE representing resource estimation error on test data for Experiment 1 (FixW), Experiment 2 (CPD), and Experiment 3 (Proposed).

Data Set	Experiment	LR	SVM	EN	LASSO	RR	NNLS	Average
Alibaba	FixW	0.84	0.78	0.72	0.77	0.83	0.94	0.81
	CPD	1.00	0.76	0.85	0.89	0.90	0.94	0.89
	Proposed	**0.55**	**0.52**	**0.49**	**0.53**	**0.51**	**0.56**	**0.53**
Bitbrains	FixW	0.61	0.85	0.50	0.59	0.55	0.65	0.63
	CPD	0.93	1.00	0.76	0.90	0.80	0.95	0.89
	Proposed	**0.53**	**0.58**	**0.48**	**0.53**	**0.51**	**0.53**	**0.53**
Materna	FixW	0.55	0.54	0.50	0.53	0.55	0.58	0.54
	CPD	1.00	0.53	0.60	0.85	0.64	0.99	0.77
	Proposed	**0.49**	**0.49**	**0.46**	**0.47**	**0.48**	**0.42**	**0.47**

all three observation window selection methods and resource estimation is selected to calculate the normalized MSE. The proposed solution yields the minimum MSE using all estimation methods. The best performing estimation method is EN for Alibaba and Bitbrains data sets, whereas NNLS outperforms in Materna data set.

To quantify the relative improvement for estimation resource utilization using the Proposed window size selection method, we consider FixW (Experiment 1) as a baseline method. Figure 5.15 shows the relative percentage of MSE for the Proposed and CPD compared to the FixW. For all estimation methods, the Proposed outperforms the FixW and the CPD to minimize the estimation error. Whereas, the FixW yields better estimation results compared to the CPD. For Alibaba data set, the proposed method produces 34%, 33,%, 32%, 31%, 38%, and 40% better estimation results for LR, SVM, EN, LASSO, RR, and NNLS estimation methods respectively compared to FixW. For Materna data set, the Proposed method gives 11%, 9%, 9%, 18%, 18%, and 27% better estimation results for LR, SVM, EN, LASSO, RR, and NNLS estimation methods respectively compared to FixW. For Bitbrains data set, the Proposed method gives 13%, 32%, 9%, 10%, 9%, and 18% better estimation results for LR, SVM, EN, LASSO, RR, and NNLS estimation methods respectively compared to FixW.

In average, we observe 29.7%, 15.7%, and 13.1% better estimation results using the proposed method for Alibaba, Materna, and Bitbrains data sets respectively. The comparison indicates that the Proposed method outperforms FixW and CPD for

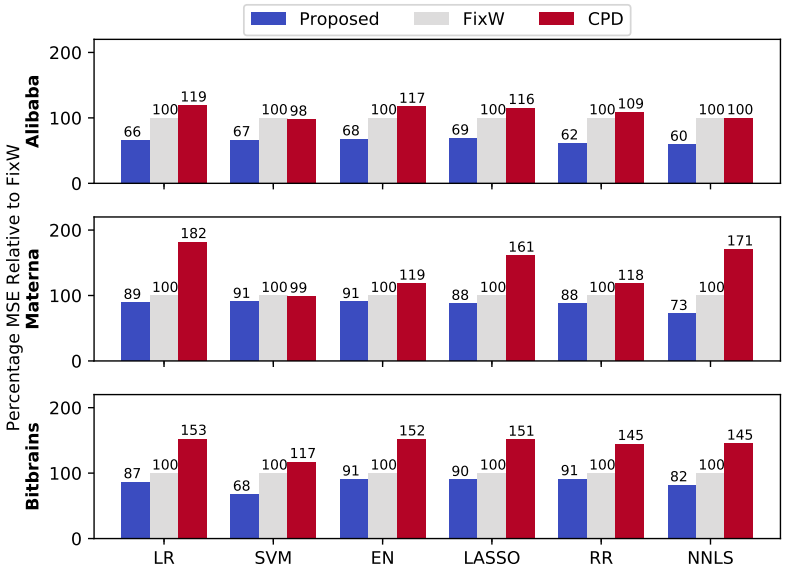


Figure 5.15: Comparison of MSE for FixW and CPD and Proposed method using different estimation methods.

identifying appropriate observation window to build estimation method with good results.

A traditional solution to identify observation windows for building estimation methods are based on a series of experiments using various fixed size observation windows manually and then determine the best observation window to use with the estimations. Whereas, an automatic adaptive observation window size can help to minimize estimating error better than fixed observation windows and also reduce the human efforts to test different observation window sizes manually. In our experimental evaluation, we evaluated six different commonly used estimation methods and showed that adaptive observation windows could yield better estimation results comparing to the fixed observation windows for all different estimation methods.

5.6 Related Work

Many researchers have recently addressed data center resource estimation. For example, a recent work by Mason et al. [76] estimated the host CPU utilization using Neural Networks trained on a fixed observation window sizes containing entire day data. Zhang et al. [135] proposed cloud workload prediction system for industry informatics based on stacked autoencoders. They use a canonical polyadic decomposition format to reduce the training time by compressing the input parameters. The author also used a fixed observation window size to train and test the proposed estimation method. Another work by Nikravesh et al. [84] proposed a predictive auto-scaling system to scale the cloud resources automatically. The proposed approach uses SVM and neural network for the different type of workloads and estimation methods are trained using a fixed observation window size. However, the work studies the effect of different observation window sizes. Yang et al. [129] used linear regression to predict the cloud workload using a fixed observation window size of 4. Davis et al. [26] proposed hardware failure prediction system for cloud data centers using Neural Networks however they also used fixed length sliding window.

All of the above-mentioned works use fixed observation windows. However, there are some contributions which use dynamic observation windows for resource estimations. For example, Dalmazo et al. [25] proposed a method to us dynamic observation windows to estimate the network traffic in the cloud using statistical techniques. The proposed approach uses the variance of the data for the current and the last observation windows and adjusts the next observation window size. Klinkenberg et al. [120] use the adaptive sliding window to forecast the time series by identifying the changes in the underlying data generation process. They started with a default initial window size and kept increasing or decreasing it minimize the forecasting error. However, the proposed approach requires an extensive search to identify appropriate window size which is not feasible for real-time processing. A recent work by Tschumitschew et al. [116] select the optimal window size for the regression problem to predict the next data point in the time series based on the ratio of drift and noise. They use larger window size if noise is stronger than drift otherwise shorter window size is used for the estimation.

The dynamic observation windows are used for a variety of different problems including physical activity recognition, industrial process optimization, mining frequent itemsets, wireless networks, health care, and renewable power generation. Wang et al [122] use adaptive time series windows with Convolutional Neural Networks to optimize the industrial process operations. Authors proposed to select different time-series windows according to the steady and unsteady states in the given historical time series observations. Ouyang et al [89, 88] use optimal window size for wind power ramp prediction by minimizing the non-ramp data in time windows. Some of the recent work shows the use of a dynamic sliding window in the health care domain. For example, Smrithy et al. [107, 82] use a dynamic sliding window with weighted moving averages to detect the anomalies in the wireless body area networks which are used in real time healthcare systems. By identify anomalies, they decrease the false alarm rate which results in increasing the reliability of the system. They decide the size of the window by comparing the variance of predecessor and current sliding window. Pérez-Solano et al [92] use adaptive window size for linear regression to synchronize time in wireless networks. They search for a window size which yields minimum Mean Square Prediction Error (MSPE). Similarly, Noor et al. [85] use the adaptive sliding window for signal segmentation which is used to recognize the transitional physical activity. The proposed method uses a default window size as a starting point and then used probability density function to expand the size of the window to capture the transitional activity with a longer duration. They keep increasing the size of the window iteratively until the probability density function reached its highest value.

Another work by Deypir et al. [27] use variable size sliding window to mine frequent itemsets. They start with an initial window size which is set by luser and then the window size is adjusted according to the rate of change in the incoming data stream. They increase the window size if no significant difference is detected and reduce the size if substantial changes are observed in the data. However, the limitation of this approach is that the size of the window becomes huge when there is no changes occurred in data. A recent work by Chou et al. [22] proposed a time series prediction system based on metaheuristic optimization of sliding windows. The purpose of the system is to forecast the stock price. However, the proposed method is computationally expensive and not feasible for real-time resource estimation.

Most of the existing works for resource estimation either use fixed observation windows or use simple statistical methods to decide the dynamic observation window sizes. However, some contributions use extensive search-based methods to identify the observation window sizes which are computationally intensive and infeasible for

real-time resource estimations. Our work reported in this paper uses a novel deep learning based window size estimation method to efficiently identify the best observation windows to train the regression models for estimating future resource utilization.

5.7 Final Considerations

The selection of appropriate historical data to train the estimation models plays an important role in prediction and accuracy can be degraded by selecting the wrong number of observations. Therefore this chapter focus on the problem of fixed size sliding windows which are used to train the prediction model. The prediction accuracy can be improved if there is a possibility to select the sliding window size. To address this issue, this chapter presents a novel method to automatically identify the number of historical observations to use for building estimation models in an adaptive way. To identify the amount of historical data to be used in training estimation model, we use multi-layer perceptron. The experimental results show that the prediction accuracy can be improved from 9% to 40% as compared to baseline methods when evaluated on different type of real resource utilization traces collected from Alibaba, Materna and Bitbrains data centers. In the future, we intend to investigate the identification of the appropriate machine learning algorithm automatically for building estimation models using the proposed adaptive observation windows.

Chapter 6

Conclusions

6.1 Main Contributions

The thesis concludes with three contributions in achieving the thesis goal. The first contribution of this thesis, presented in Chapter 3 consists of data center modeling using Markov Chain Models (MM) to reduce the storage space and bandwidth utilization of data centers' telemetry data. Data centers generate a lot of telemetry data which is used for many purposes including resources management, analytics and optimization. However, the size of telemetry data grows dramatically and considerably increases the storage space and bandwidth utilization within the data center. In this contribution, we proposed a Markov chain-based method to reduce telemetry data to minimize bandwidth utilization and storage space required to store it within the data center. Our solution outperforms the baseline method based on Polynomial Regression (PR) method to reduce and regenerate the telemetry data. We extensively evaluated the effect of batch sizes, number of states in MM and polynomial degrees in PR. We observed that a larger batch size effectively reduces data but the reconstruction accuracy is lower. Therefore, we identified a batch size between 16 to 64 is appropriate to use for data reduction with better reconstruction accuracy. Our experimental evaluation shows that Polynomial Regression-based method required more storage space as compared to Markov Chain Model-based method due to the high precision of coefficients. We also observed that 95.33% storage space and 75% bandwidth utilization could be reduced with 92% accuracy using the proposed solution.

The second contribution of this thesis is to explore the modeling techniques for improving the telemetry prediction accuracy and in this contribution, we develop a novel method to achieve the required goal. Building new methods for estimating resource utilization in data centers is an active and challenging problem, as most of the state-of-art techniques are based on specific machine learning methods, able to adjust to particular scenarios, but ineffective on extremely diverse environments. Therefore, we present a novel approach to adaptively and automatically identify the most appropriate machine learning method to be used for predicting future resource utilization, given recent observations of such resources. In our proposed methodology, we use Random Decision Forest classifiers to determine, from a set of available machine learning techniques, which one is most appropriate for predicting resources on a next time interval, having monitored the previous one. The RDF is trained on the statistical features extracted from historical observations and samples of the best method identified for each time window. Our selected available methods include several techniques used in the current state of the art, as regression methods, neural networks, statistical learning, and bayesian approaches. The proposed method is evaluated on real traces collected from Alibaba and Bitbrains data-center monitoring data sets, and our proposed approach can improve prediction accuracy from 6% to

27% over current methodologies. We also focused on the importance of monitoring time window sizes when modeling and predicting and evaluated different sizes. We found that 60 minutes of historical resource utilization observation can effectively be used to build the prediction model to estimate future resource utilization.

The third contribution of this thesis focuses on the development of a novel method to dynamically identify the sliding window size which is used to train the prediction models. The existing state-of-art methods available up-to-date use fixed size sliding windows to train estimation models. This traditional method does not consider the local changes and patterns in the resource usage of data centers and always using a fixed sized recent data points to use for training machine learning models which do not produce accurate future estimations. In this contribution, we devise a method to automatically limit the observation window size to use for building estimation models at every estimation step adaptively. Our proposed solution uses a multi-layer perceptron to identify the observation window sizes to be used by estimation methods for build models with improved estimation results to multiple baseline approaches. The proposed solution is evaluated on real resource utilization traces collected from Alibaba, Materna and Bitbrains data centers. Our proposed method can improve prediction accuracy from 9% to 40% over current methodologies. We conclude that the proposed system can help to identify the appropriate window size for each specific scenario over time for building estimation models with higher accuracy compared to the existing state-of-the-art baseline techniques.

6.2 Topics for Further Research

We believe that these contributions lead to many other open research areas which could be interesting to explore as future work. Therefore, in this section, we present some of the interesting future directions for the work done in this thesis.

- As part of this thesis, we presented the telemetry reduction framework based on Markov Chain Models. In this contribution, we created MM models with predefined number of states and for fixed size batches, While this approach has proved to be good for reducing telemetry data, there is still room for improvement and this contribution can be extended by adaptively identifying the batch size and the number of states in MM to further reduce space and increase the reconstruction accuracy. Another option to improve this contribution would be using one Markov Model per metric at the data center level for recurring workloads having similar resource usage requirements.

- The adaptive model selector discussed in chapter 4 of the thesis presented the methodology which can help to identify the appropriate machine learning methods for each specific scenario over time. In this contribution, we only cover CPU to demonstrate the effectiveness of the proposed model. However, this contribution could be extended by adding other telemetry streams such as memory, network. I.O e.t.c. Moreover, the work focus on simple and classical machine learning techniques however this contribution can also be extended in another future direction by investigating the deep learning methods to model the data

center's telemetry streams which is not covered in this thesis.

- The adaptive window size selector framework discussed in chapter 5 presented the method which adaptively selects the window size for the prediction model. In this contribution, these prediction models are used for CPU telemetry data only however, this contribution can be extended by adding other telemetry streams such as I.O. memory and network. This work can also be extended by integrating this contribution in the second contribution which means the identification of the appropriate machine learning algorithm automatically for building estimation models using the proposed adaptive observation windows.

6.3 List of Publications

The contributions of this thesis appear in the following publications.

- Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, David Carrera, "Real-time Data Center's Telemetry Reduction and Reconstruction Using Markov Chain Models," IEEE Systems Journal
- Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, David Carrera, "Adaptive Prediction Models for Data Center Resources Utilization Estimation," IEEE Transactions on Network and Service Management
- [Major revision submitted] Shuja-ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, David Carrera, "Adaptive Sliding Windows for Improved Estimation of Data Center Resource Utilization," Future Generation Computer Systems

The contributions from preliminary work and collaboration appear in the following publications.

- Shuja-ur-Rehman Baig, Marcelo Amaral, Jordà Polo and David Carrera, "Performance Characterization of Spark Workloads on Shared NUMA Systems," 2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (Big-DataService), Bamberg, 2018, pp. 41-48.
- [Submitted] Kiyana Bahadori, Shuja-ur-Rehman Baig, Alberto Gutierrez-Torre, Waheed Iqbal, Tullio Vardanega, Josep Lluis Berral, David Carrera, "Towards Modeling Traffic Data on Edge Infrastructures using Distributed Deep Learning," Future Generation Computer Systems

Appendix A

Performance Characterization of Spark Workloads on Shared NUMA Systems

In this chapter, we presents the impact of NUMA topology on Spark workloads as the preliminary work of PhD thesis. Nowadays, due to the growth in the number of cores in modern processors, parallel systems are built using Non-Uniform Memory Architecture (NUMA), which has gained wide acceptance in the industry, setting the new standard for building new generation enterprise servers. These processors can be connected to large amounts of physical memory, in the range of up to a couple of terabytes for the time being. This opens an enormous range of opportunities for runtimes and applications that aim to improve their performance by leveraging low latencies and high bandwidth provided by RAM. The result is that today there are several examples of applications that have started pushing the *in-memory computing* paradigm to accelerate tasks. To deliver such a large physical memory capacity, sockets in NUMA systems are connected through high performance connections and each socket can have multiple processors with its own memory. A process running on a NUMA system can access the memory of its own node as well remote node where the latency of memory accesses on remote nodes is significantly high compared to local memory accesses [13]. Ideally, memory accesses are kept local in order to avoid this latency and contention on interconnect links. Moreover, the bandwidth of memory accesses to last-level caches and DRAM memory also depends on the access type that is local or remote. From the NUMA perspective, people want to learn whether NUMA topology can meet the challenges of in-memory computing frameworks, and if not, what kinds of optimizations are required. At the same time, as the adoption of Big Data technologies becomes the norm in many scenarios, there is a growing need to optimize them for modern processors. Spark [132] has gained momentum over the last few years among companies looking for high performance solutions that can scale out across different cluster sizes. To achieve a good performance for in-memory computing frameworks on a NUMA system, there is a need to understand the topology of the interconnect between processor sockets and memory banks. Additionally, while a NUMA architecture can provide very high memory capacity to the applications, it also implies the additional complexity of letting the Operating System take care of many critical decisions with respect to where data is physically stored and where are processes accessing that data placed. This fact may have no impact for many applications that are not memory intensive, whereas memory-bound applications can be seriously impacted by these decisions in terms of performance. This chapter explores how Spark-based in-memory computing workloads are impacted by the effects of NUMA architecture, how different Spark configurations result in changes in delivered performance, how the characteristics of the applications can be used to predict

workload collocation conflicts, and how to leverage memory-consumption patterns to smartly co-locate workloads in scale-up nodes.

A.1 Background

A.1.1 Apache Spark

ıle in-memory data
the file system as
ıry across phases in
131] which are im-
details of distribu-
des a programming
ii) actions. Trans-
nch a computation
distributed storage
s a directed acyclic
contain pipelined
into tasks. A task
ıe tasks of a stage
ads which are used

which is 2-way 12-
ch core able to run

NUMA node) with
e Power8 processor
ache, a store-in L2
KB, 512KB, and 8
s of eight external

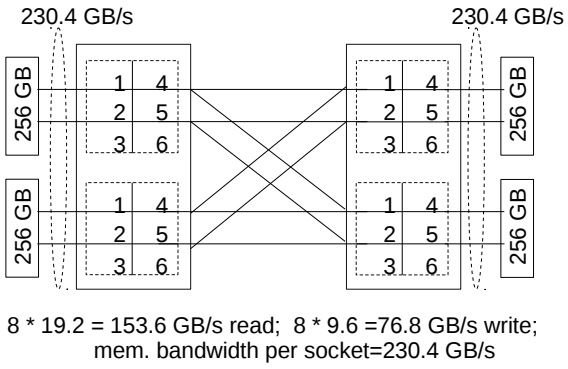


Figure A.1: Power8 NUMA architecture

Because of the main memory is connected to the processor using separate links for reading and write operations, with two links for memory reads and one link for memory writes, the system has asymmetric read and write bandwidth. The connection between the processor with the memory is composed of 8 links, with a link offering 9.6

parameter	value
spark.serializer	org.apache.spark.serializer.KryoSerializer
spark.rdd.compress	FALSE
spark.io.compression.codec	lzf
storage level	MEMORY_AND_DISK
spark.driver.maxResultSize	2 gb
spark.driver.memory	5 gb
spark.kryoserializer.buffer.max	512m

Table A.1: Spark configuration parameters

GB/s write and 19.2 GB/s read bandwidth. Therefore, the system has in total four NUMA nodes, 192 virtual cores, 1 TB of RAM and a total of 230.4 GB/s of sustainable memory bandwidth per socket, as illustrated in Figure A.1. For the software stack, the machine is configured with a Ubuntu 16.10, kernel 4.4.0-47-generic, IBM java version 1.8.0 and Apache Spark 1.6.1.

A.2 Methodology

This section describes how the study on the impact of NUMA topology on in-memory data analytics workloads has been performed, as well as the rationale behind the experiments evaluated in the following sections.

A.2.1 Workloads

The experiments presented in this chapter are based on Spark-Bench [69], which is a benchmark suite developed by IBM and widely tested in Power8 systems. From the range of available workloads provided by the benchmark, Support Vector Machines (SVM), PageRank, and Spark SQL RDDRelation have been selected for the evaluation. These workloads are well-known in the literature, and combine different characteristics to cover a large range of possible configurations. Dataset size for SVM, SQL and PageRank is 47, 24, and 17 GB and number of partitions are 376, 192 and 136 respectively for all experiments.

A.2.2 Experimental Setup

Since the goal of this chapter is to evaluate the performance of Spark workloads on NUMA hardware, all the experiments are conducted in a single machine; the characteristics of the machine's architecture are described in Section A.1.2. For simplicity, Spark is configured in the standalone mode [5], To control the number of cores, memory, and the number of executors of each worker, the parameters `SPARK_WORKER_CORES`, `SPARK_WORKER_MEMORY`, and `SPARK_EXECUTOR_MEMORY` [5] are used, respectively. All the other parameters and values used to configure Spark, during the experiments execution described later in this chapter, are summarized in Table A.1.

Hardware counters have been used to collect most real-time information from experimental executions, using the perfmon2 [93] library. Memory bandwidth is calculated based on the formula defined in [38]. For CPU usage, memory usage, and context switches, the vmstat tool has been used. To collect information about NUMA memory access, the `numastat` is used. Finally, the `numactl` command has been used to bind a workload in a set of CPUs or in memory regions (e.g. in a NUMA node); the `n`B nomenclature is used to describe different binding configurations, where `n` is

wem	tma	tsw	core per worker									
			1	2	3	4	6	8	12	16	24	48
250	1000	4	4	8	12	16	24	32	48	64	96	192
125	1000	8	8	16	24	32	48	64	96	128	192	
83	996	12	12	24	36	48	72	96	144	192		
62	992	16	16	32	48	64	96	128	192			
41	984	24	24	48	72	96	144	192				
31	992	32	32	64	96	128	192					
20	960	48	48	96	144	192						
15	960	64	64	128	192							
10	960	96	96	192								
5	960	192	192									

Table A.2: Experiment 1: Evaluated software configurations (*wem* is worker and executor memory; *tma* is total memory allocated; and *tsw* is total Spark workers)

the number of assigned NUMA nodes, e.g. *1B* for workloads bound to 1 NUMA node, *2B* for workloads bound to 2 NUMA nodes, etc.

A.3 Experiment 1: Workload Characterization

This experiment consists of a performance characterization of Spark workloads, changing the configuration parameters of Spark itself and observing the impact of different configurations in terms of completion time and resource consumption. The goal is to find which configurations lead to optimal performance, e.g. which number of cores per Spark worker, and/or the number of workers per application. Since this experiment aims at defining the software configuration, there is no other kind of hardware tuning involved; the OS performs its default resource allocation decisions. More specifically, this experiment analyzes the effect of the software configuration in the resource usage intensiveness and possible speedups for the workloads described in Section A.2.1.

As described in Section A.1.2, the machine used for the experiments has 4 NUMA nodes, 192 virtual cores, and 1TB of main memory. Thus, this experiment varies the number of cores, workers, and memory up to the total amount of resources (cores and memory) available in this machine. Table A.2 describes all combinations of the amount of resources allocated to all workloads in this experiment. The amount of memory ranges from 5 up to 250GB per worker, the number workers vary from 4 up to 192 workers. Depending on the number of workers, the number of virtual cores ranges from 1 up to 192. If the total number of workers is 192, each one will have only one virtual core. Note that, by creating this matrix of experiments, we want to see at which configuration, the operating system produce optimal results for each workload type in terms of completion time. We assign memory to Spark worker and executor by dividing 1000 by a total number of workers in the experiment and take the integral part only; thus, the amount of memory ranges from 960GB to 1000GB. Some amount of memory is intentionally left for the OS and other processes (e.g spark driver, master) to avoid the slowdown effects not related to NUMA.

In Spark, a software configuration defines the number of workers, the number virtual cores and the amount of memory per worker that is assigned to a specific workload. These software resources need to match the hardware configuration of the node used to run the workloads. But not all applications can take advantage of an increasing amount of resources and therefore it is not always the case that one single

workload	total workers	core per worker	total cores allocated	worker executor memory	total memory allocated	Execution Time (sec)
SVM	24	8	192	41	984	323.71
SQL	4	12	48	250	1000	206.82
PageRank	12	8	96	83	996	748.08:

Table A.3: Experiment 1: Best configuration when optimizing for completion time

software configuration optimizes the performance of a Spark Workload for a given hardware setup.

Table A.3 summarizes the optimal configurations that found for the three workloads considered in this experiment. As it can be seen, every workload achieves maximum performance using a different software configuration, being SVM the application that can take advantage of more threads in parallel, followed by PageRank and finally SQL. It is remarkable that even configurations with a similar number of cores allocated tend to deliver different performance for similar configurations of number of workers and number of cores per worker. For instance, SQL works best with fewer workers and more cores per worker and SVM gets the best performance when more workers and fewer cores per worker are assigned. This is due to SQL is more impacted because of thread locks and cache contention than the SVM. Hence, SQL benefits from fewer threads competing for resources. Additionally, because the JVM includes additional overheads (e.g. garbage collection), more layers for resource management and memory indexing, it is not beneficial to have several workers with only one virtual core.

To explain the root cause of the performance delivered by the different configurations, Tables A.4 , A.5 and A.6 also show the executions times in seconds obtained for SVM , SQL, and PageRank respectively when using all combinations of software configurations, but in this case, we color each configuration according to the relative performance delivered compared to the optimal configuration found for that particular workload. Based on this property we classify configurations in different groups:

- *Within 10% of optimal*: configurations for which completion time is very close to the best execution time observed for that particular workload.
- *Low CPU Usage*: configurations for which CPU usage is clearly below the observed CPU usage for the workload. These configurations use a too low number of cores or workers that are not enough to fully utilize the available compute resources and produce optimal results.
- *High CPI and Context Switches*: executions where cycles per instruction (CPI) and context switches are greater than the observed values for the optimal configuration. This is due to more executors which execute more threads to process the tasks. Moreover, executors need to communicate with each other and also with drivers. Remote memory access also impacts the CPI since it requires more CPU cycles to be performed than local access. This leads to increase in communication overhead. So in result, we see an increase in context switches and CPI.
- *High L3 misses*: configurations where L3 cache misses are greater than in the optimal configuration. This group is only defined for SVM as it is the only workload for which this behavior was observed.

- *Low Memory Bandwidth*: configurations where memory bandwidth usage is less than the observed value for the optimal configuration.
- *Require more investigation*: configurations where values of metrics are within the range of optimal region but the completion time is outside of 10%. The experiments in this region require further investigation and it could not be determined so far the reason for the performance differences with the optimal configuration.

w	core per worker									
	1	2	3	4	6	8	12	16	24	48
4	1437.95	1018.4	816.87	698.34	597.13	515.9	501.78	464.26	472.28	375.2
8	759.16	555.26	478.23	411.91	366.39	357.54	347.49	359.18	360.48	
12	531.9	422.6	420.24	382.46	386.72	332.44	353.68	339.51		
16	458.58	412.85	385.79	352.69	347.22	333.63	336.26			
24	413.1	394.72	371.6	358.46	354.17	323.71				
32	405.16	389.16	369.81	361.36	341.53		Within 10% of optimal			
48	442.8	427.85	398.61	370.78			Low CPU usage			
64	546.11	522.3	569.83				High CPI and Context Switches			
96	1118.77	922.01					High L3 misses			
192	1980.92						Require more investigation			

Table A.4: SVM completion time (seconds) groups

w	core per worker									
	1	2	3	4	6	8	12	16	24	48
4	1148.69	616.46	427.64	338.64	288.48	231.35	206.82	229.65	235.82	238.99
8	590.22	335.51	264.95	255.86	220.93	209.33	259.7	236.06	220.27	
12	426.48	270.56	244.68	229.2	217.48	223.57	296.5	228.74		
16	375.65	288.92	242.9	247.26	241.71	245.1	240.67			
24	347.14	284.32	264.94	254.9	282.7	246.8				
32	347.65	293.96	268.68	287.77	262.32					
48	328.69	299.43	285.76	282.98			Within 10% of optimal			
64	324.99	307.44	307.54				Low CPU usage			
96	349.75	355.28					High CPI and Context Switches			
192	591.11						Require more investigation			

Table A.5: SQL completion time (seconds) groups

w	core per,worker									
	1	2	3	4	6	8	12	16	24	48
4	4771.33	2028.77	1532.34	1188.68	1016.5	1000.84	2358.19	2207.09	1733.52	3186.17
8	2517.32	1145.33	997.91	902.02	920.09	861.18	816.06	1148.35	1319.35	
12	1580.25	980.71	911.1	785.84	788.52	748.08	1028.29	1010.76		
16	1379.76	921.87	871.61	862.69	766.9	925.91	877.9			
24	1175.22	909.71	866.08	843.5	780.68	812.44				
32	1085.66	875.52	907	1095.11	880.61		Within 10% of optimal			
48	996.54	858.22	760.27	767.63			Low CPU usage			
64	1183.04	1143.43	912.03				High CPI and Context Switches			
96	1447.05	1142.42					Low Memory Bandwidth			
192	2918.32						Require more investigation			

Table A.6: PageRank completion time (seconds) groups

The second objective of this experiment was to characterize the CPU, Memory Footprint and Memory Bandwidth demands of each one of the workloads of study. For this purpose, we monitored the execution of the workloads when the optimal software configuration was in use and plotted the average resource consumption in Figure A.2. As it can be seen, results show that memory usage is 457.7 GB, 364.3 GB and 329.4 GB for SVM, SQL and PageRank respectively and shows the average usage of user CPU time and memory bandwidth for these workloads when the optimal software configuration is in use. As it can be observed, SVM is constrained by the high CPU usage, reaching around 80% for the user CPU time only, that when added

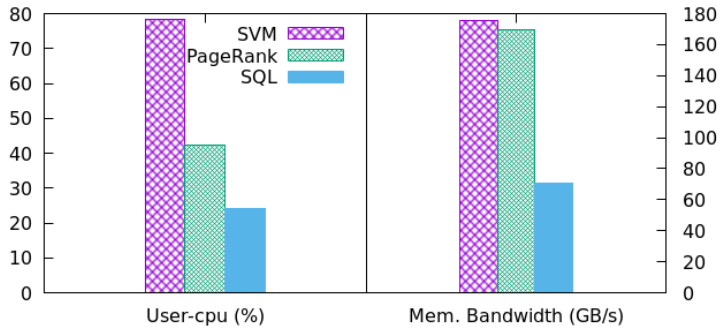


Figure A.2: Experiment 1: CPU Usage (percentage) and Memory Bandwidth (GB/s) for optimal configuration

to the system and wait CPU times tops to about 100% CPU usage, which is the actual performance bottleneck.

SQL is a more interesting case because CPU and Memory Bandwidth usage are really low for the fastest configuration, and no other resource is apparently acting as a bottleneck. In practice, what is avoiding the total CPU usage to go higher is the fact that the number of threads that are spawn in this configuration (only 48) is well below the number of hardware threads offered by the system. Therefore, only a third of the hardware threads are in use and that is why the average CPU utilization is shown to be low: several hardware threads are idle. Intuition would say that increasing the number of threads would increase the performance, but in practice what is observed in the logs of other experiments is that as soon as the number of threads goes higher the memory bandwidth dramatically increases, quickly becoming the bottleneck at many stages during the execution. The bottom line is that the OS is not able to correctly manage the threads for this workload, creating memory access patterns that saturate the memory links of the P8 processor.

Finally, PageRank is in the same situation: the optimal configuration involves 96 software threads only, while the system offers 192 hardware threads. In practice, what it means is that the reported CPU utilization is low. Intuition again would point in the direction of increasing the number of software threads, but when that direction is taken, logs show that the additional software threads start competing for memory bandwidth because they exhibit worse memory access patterns, and saturate the memory links.

In summary, this experiment has shown two cases in which not all hardware threads could be exploited because in that case the memory access patterns across NUMA nodes were hitting a memory bandwidth bottleneck. This is an interesting result because opens a door to smart workload collocation strategies that will be explored in the following experiments.

A.4 Experiment 2: Binding to NUMA Nodes

Allocating more NUMA nodes to a workload has the potential to increase resources (such as memory bandwidth, CPU, and memory) and possibly lower cache contention (due to the availability of additional cores and cache), but it can also involve a trade-off: using remote memory accesses and dealing with bus contention, which can lead to slowdowns in some scenarios. Binding, in this context, means Spark processes (master and workers) will only have access to the resources (cores and memory) of a particular set of NUMA nodes. While the previous experiment selected the optimal

n(x) to n(y)	SVM	SQL	PageRanK
1B → 2B	1.44x	1.73x	1.79x
1B → 4B	2.07x	2.63x	2.64x
2B → 4B	1.44x	1.52x	1.47x
3B → 4B	1.1x	1.11x	1.13x
4NB → 4B	1.15x	1.07x	1.25x

Table A.7: Experiment 2: Speedup (B=Node with binding, NB= Node without binding)

software configuration without binding, allowing the operating system to make all decisions, this experiment selects the optimal software configuration when binding all 4 nodes (4B). Results are also compared to the previous experiment so as to evaluate the impact of binding.

In the previous experiment, the OS was responsible for allocating all the resources, in this experiment we enforce decisions to manually bind the workloads across the NUMA nodes. The main motivation of performing the manual binding is to mitigate the limitations defined in the previous experiments. Manually binding the workloads can exploit better load balancing, minimize thread migrations and remote memory access. In order to verify those assumptions, we evaluate the completion time of all workloads for different binding configurations and compare with non-binding approaches (the default allocation from OS). As explained in Section A.4, the workloads are bound in one NUMA node up to 4 (1B, 2B, 3B, 4B). The results of the optimal software configuration considering the different number of NUMA nodes is shown in Table A.7. It also shows the comparison of the manual binding with four NUMA nodes versus the default OS resource allocation, labeled as NB.

The optimal configurations are 24 cores per worker and 1 worker per node for SQL, SVM, and PageRank when we bound workloads to one NUMA node (1B). In case of 2B, the optimal configurations are 8 cores per worker and 3 workers per node for SQL, 6 cores per worker and 6 workers per node for SVM and 4 cores per worker and 6 workers per node for PageRank. Similarly, in case of 3B, the optimal configurations are 8 cores per worker and 2 workers per node for SQL, 6 cores per worker and 4 workers per node for SVM and 3 cores per worker and 6 workers per node for PageRank. The optimal configurations in case of 4B are 12 cores per worker and 1 worker per node for SQL, 8 cores per worker and 3 workers per node for SVM and 6 cores per worker and 3 workers per node for PageRank.

The results of this experiment, as summarized in Table A.7, show a significant speedup when comparing manual binding versus the OS allocating the resources, but not for all workloads. In the case of SVM, SQL and PageRank, the speedups are x1.15, x1.07 and x1.25, respectively. This is related to what was discussed in the previous experiment, the main bottleneck is related to resource contention, most especially the memory bandwidth. Because of manual binding minimizes remote memory access and local memory access has higher memory bandwidth and lower latency, some workload benefit from that. However, this improvement impacts different each workload what depends on the workload memory access pattern.

The results of this experiment also show that how applications scale with more NUMA nodes. Considering following formula (current node(s) + new node(s) / current node(s)), the theoretical speedup of allocating one new NUMA node to application with one current node would lead to a speedup of 2x. The results actually show different speedups for all workloads. PageRank is the workload that scales better, with 1.79x from one (1B) to two (2B) NUMA nodes. In the case of SVM, speedup

			w1	w2	w1	w2	w1	w2
n-n	**w1**	**w2**	**w/n**	**w/n**	**c/w**	**c/w**	**w+e**	**w+e**
2-2	SQL	SVM	3	6	8	6	83	41
2-2	SQL	PR	3	6	8	4	83	41
2-2	SVM	PR	6	6	6	4	41	41
1-3	SQL	SVM	1	4	24	6	250	62
1-3	SQL	PR	1	6	24	3	250	41
1-3	SVM	PR	1	6	24	3	250	41
3-1	SQL	SVM	2	1	8	24	125	250
3-1	SQL	PR	2	1	8	24	125	250
3-1	SVM	PR	4	1	6	24	62	250

TABLE A.8: Configurations for different co-scheduling workload combinations; (n-n=NUMA node combination; w1=workload-1; w2=workload-2; w/n=worker per node; c/w-=core per worker; w+e=worker and executor memory; PR=PageRank)

for 1B-2B is only 1.44x. SQL is in between them with a 1.73x speedup for 1B-2B. In the case of increasing from 3 to 4 NUMA nodes (3B to 4B), the theoretical speedup would be 1.33x and results show the speedup in range of 1.1x to 1.13x.

A.5 Experiment 3: Workload Co-scheduling

This final experiment explores the benefits of workload co-location and process binding (cores and memory) as a mechanism to improve system throughput and increase resource utilization. Workload co-location is well-known to possibly slow down interference-sensitive applications, however, the impact of NUMA co-scheduled Spark workloads are still not completely understood. This experiment, therefore, evaluates the performance impact on workloads when sharing the same machine, that is when workloads are co-located. In order to do so, a defined set of workload combinations among the four NUMA nodes is provided. This experiment aims at evaluating the trade-offs between sharing NUMA nodes without binding (with the OS using simple resource allocation policies), versus binding the workloads to isolate their interference from each other.

The baseline for this experiment is based on the optimal configurations from Experiment 2. Table A.8 describes all the workload co-location scenarios along with their software configurations. For all configurations evaluated in this experiment, only two workloads run at the same time. First, the workloads are equally spread to the four NUMA nodes. That is, each workload is allocated to two NUMA nodes, labeled as 2B-2B. Secondly, uneven configurations are evaluated, such as 1B-3B and 3B-1B, meaning one of the workloads is bound to 1 NUMA node and the other one to 3 NUMA nodes. Note that all configurations are executed with binding on NUMA nodes according to the description in Table A.8, but also without binding where the OS allocates all shared resources.

For evaluation purposes, more than one workload is executed at a time in the system under test. As each workload exhibits different job duration characteristics, we evaluate the effects of co-location on a continuous workload environment. In particular, for any pair of workloads that is evaluated, we run both of them in a continuous loop of 90 minutes, from which the first 15 minutes are taken as a warm-up period and

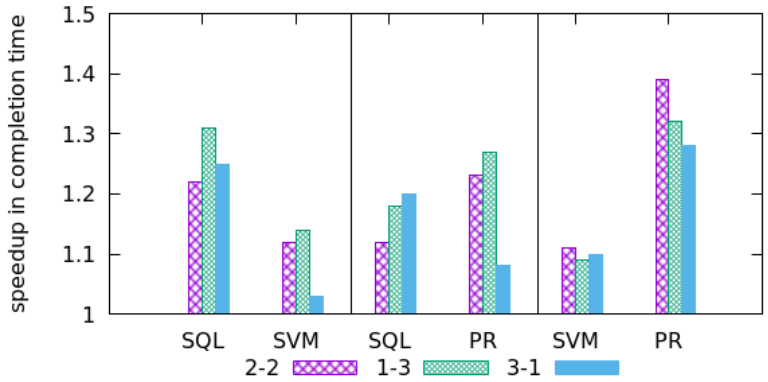


Figure A.3: Experiment 3: Completion time speedup (60 minutes interval; binding vs non-binding(OS default allocation))

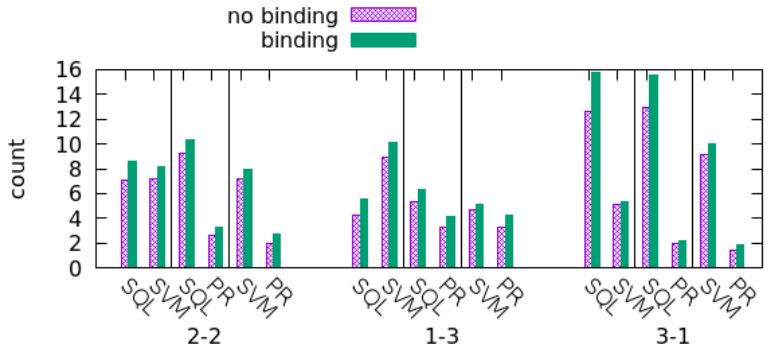


Figure A.4: Experiment 3: Number of executions (60 minutes interval)

the final 15 minutes as a cool-down process. The central 60 minutes of the experiment are used to measure the performance of the system. During that period we collect system telemetry as well as account for how many loops every workload managed to perform.

The first configuration evaluated makes an even distribution of the NUMA nodes across the two workloads under test. We label this test *2B-2B* (2 NUMA nodes with process binding). Each partition runs a different workload. Each workload is configured with the optimal 2B configuration found in the previous experiment. We repeat the process with the *1B-3B* configurations (1B for 1 NUMA node with binding, and 3B for 3 NUMA nodes with binding), in which one workload gets assigned one NUMA node while the other gets allocated the other three nodes.

As it was mentioned before, to provide a baseline for these experiments we also run the same workload co-location experiments, maintaining in every case the software configuration, but in this case without performing any process binding operation and therefore allowing the Operating System to freely schedule processes, threads and memory pages across NUMA nodes. In all cases we executed all combinations of SQL-SVM, SQL-PageRank, SVM-PageRank.

Figure A.3 shows the speedup in execution time when binding over non-binding configurations. As it can be observed, in all cases the process binding configurations outperformed the non-binding setups, providing a significant improvement that in some cases reached a 1.39X factor. Figure A.4 shows the number of executions during the 60 minute interval for all workloads, and as it can be observed, binding always outperforms non-binding. To explain the reason for this improvement, we looked at different system metrics.

In this section, we include the results for the number of CPU Context Switches and the amount of remote memory accessed per workload , as summarized in Figures A.5 and A.6 respectively.

We found that there is an increase of 36-43% in context switches when experiments are executed without binding. When processes are not bound to specific NUMA nodes, the OS is in charge of all placement decisions, which includes reactive migrations to balance the load. In a case of remote memory access, there is an increase of 80-91.93% when the same experiments are executed without binding. So in conclusion, we show that memory binding reduces the amount of remote memory access and this leads to less interconnect traffic (because of less memory transfer and cache-coherence) and contention on inter-links for Spark workloads. Moreover, CPU binding reduces the number of context switches, which also helps improve completion time for experiments.

A.6 Related Work

Although there have been several research efforts to investigate and mitigate the impact of NUMA on workload performance, this topic is still gaining traction in the literature in recent years [65], [12], [75], [64], [1], [94] [30], [31], [95]. These works characterize some key sources of workload performance degradation related to NUMA (such as additional latencies and possibly lower memory bandwidth because of remote memory access and contentions on the memory controller, bus connections or cache contention), and propose OS task placement strategies for mitigating remote memory access. But the characterization of Spark workloads on IBM Power8 systems and placement strategies for co-scheduled application is still roughly understood.

For example, [12] has characterized the NUMA performance impact related to remote memory access introduced by the OS when performing task re-scheduling or

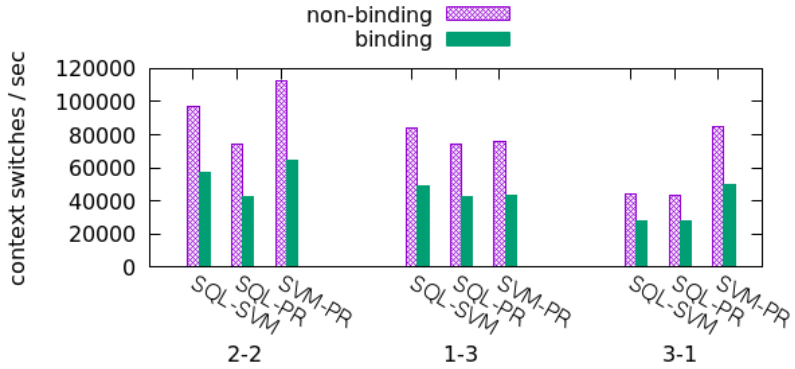


Figure A.5: Experiment 3: Context switches per second (60 minutes interval)

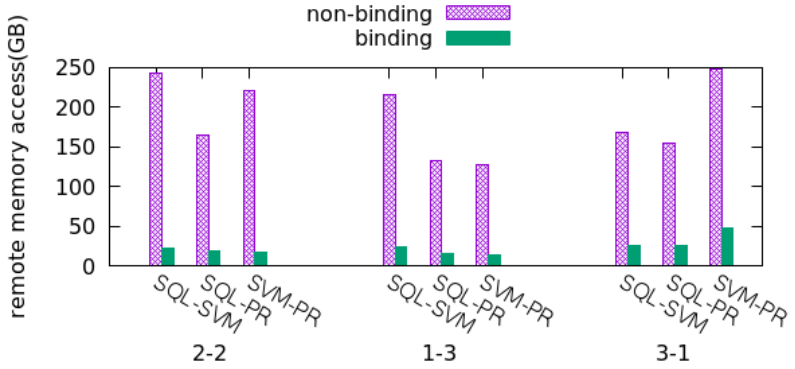


Figure A.6: Experiment 3: Amount of remote memory access in GB (60 minutes interval)

load balancing. While this work proposed an effective approach to mitigating remote memory access and cache contention, it is not application-driven and does not have a holistic view of all applications to define efficient workloads co-scheduling on NUMA systems. In our work, we demonstrate the potential benefits from manual binding strategies when co-scheduling multiple workloads on NUMA systems.

Another example is the work of [134], where the authors characterized the performance impact of NUMA on graph-analytics applications. They present an approach to minimize remote memory access by using graph-aware data allocation and access strategies. While this work presents an application-driven investigation, it lacks the analyze of memory-intensive Spark workloads and workload collocation.

The most related works to our work are the [6] and [7]. In the former, the authors quantify the impact of data locality on NUMA nodes for Spark workloads on Intel Ivy Bridge server. In the later, the authors evaluate the impact of NUMA locality on the performance of in-memory data analytics with Spark on Intel Ivy Bridge server. In both papers, they run benchmarks with two configurations a) Local DRAM b) Remote DRAM. In Local DRAM, they bound Spark process to processor 0 and memory node 0 and in Remote DRAM, they bound the Spark process to processor 0 and memory node 1. Then, they compare the results to evaluate the performance impact of NUMA. While those works present a detailed performance characterization of Spark workloads on NUMA systems on an Intel Ivy Bridge server, the NUMA performance characterization of IBM Power8 systems is still not understood. Moreover, their work

does not present the NUMA impact of optimally binding the workloads versus leaving the OS allocating the resources. Also, they do not evaluate the performance benefits of performing manual binding for co-scheduled Spark workloads as we present in this paper.

A.7 Conclusions

In-memory computing is becoming one of the most popular approaches for real-time big data processing as data sets grow and more memory capacity is made available to popular runtimes such as Spark. To deliver large physical memory capacity, modern processors feature Non-Uniform Memory Architectures (NUMA). In NUMA systems, multiple sockets are connected through high-performance connections. Each socket can have multiple processors with its own memory. A process running in a NUMA system can access the memory of its own node as well the remote node, where the latency of memory accesses is higher. It is thus important to understand how the hardware topology of a particular NUMA architecture affects the performance of in-memory computing applications. This paper analyzed the behavior of different memory-intensive Spark workloads on the IBM Power 8 NUMA processor.

Large sets of experiments were executed to evaluate several Spark workloads, and the results demonstrated that workload colocation is a smart strategy to improve resource utilization for memory-intensive workloads placed in modern NUMA processors. This conclusion is supported by the fact that the experiments showed that different kinds of Spark workloads require different software configurations to produce optimal results and that optimal configurations are commonly unable to use all available hardware resources. Highly concurrent configurations produce undesired memory access patterns across NUMA nodes that push to the limit the existing memory bandwidth, making co-scheduling a good choice. Additionally, the experiments provided insight on the existing trade-off between sharing NUMA nodes and isolating workloads; optimal configurations when binding to a different number of NUMA nodes change significantly depending on the workload and the number of nodes used. There is one constant though: when executing a workload in a single NUMA node, using a single Spark worker produces the optimal results for all workloads. The obtained results show that binding spark processes to particular NUMA nodes can speed up the completion time of co-located workloads up to 1.39x at maximum due to less interconnect traffic, less remote memory access, and less context switches and CPI.

Bibliography

[1] Jeongseob Ahn et al. “Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources.” In: *HotCloud*. 2012.

[2] Mohammad Aldossary et al. “Energy-aware cost prediction and pricing of virtual machines in cloud computing environments”. In: *Future Generation Computer Systems* (2018).

[3] *Alibaba Cluster log*. URL: https://github.com/alibaba/clusterdata.

[4] Marcelo Amaral, Jordà Polo, David Carrera, et al. “Performance characterization of spark workloads on shared NUMA Systems”. In: *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE. 2018, pp. 41–48.

[5] *Apache Spark Standalone mode*. URL: https://spark.apache.org/docs/1.6.1/spark-standalone.html.

[6] Ahsan Javed Awan et al. “Architectural impact on performance of in-memory data analytics: Apache spark case study”. In: *arXiv preprint arXiv:1604.08484* (2016).

[7] Ahsan Javed Awan et al. “Node architecture implications for in-memory data analytics on scale-in clusters”. In: *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. ACM. 2016, pp. 237–246.

[8] Donald J Berndt and James Clifford. “Using dynamic time warping to find patterns in time series.” In: *KDD workshop*. Vol. 10. 16. Seattle, WA. 1994, pp. 359–370.

[9] M. Z. A. Bhuiyan et al. “Content-Centric Event-Insensitive Big Data Reduction in Internet of Things”. In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–6.

[10] Robert Birke, Lydia Y Chen, and Evgenia Smirni. “Usage patterns in multitenant data centers: A temporal perspective”. In: *Proceedings of the 9th international conference on Autonomic computing*. ACM. 2012, pp. 161–166.

[11] *Bitbrains Cluster log*. URL: http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains.

[12] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. “Contention-aware scheduling on multicore systems”. In: *ACM Transactions on Computer Systems (TOCS)* 28.4 (2010), p. 8.

[13] Sergey Blagodurov et al. “A Case for NUMA-aware Contention Management on Multicore Systems”. In: *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC’11. Portland, OR: USENIX Association, 2011, pp. 1–1. URL: http://dl.acm.org/citation.cfm?id=2002181.2002182.

[14] Martin Burtscher and Paruj Ratanaworabhan. "FPC: A high-speed compressor for double-precision floating-point data". In: *IEEE Transactions on Computers* 58.1 (2009), pp. 18–31.

[15] Rodrigo N Calheiros et al. "Workload prediction using ARIMA model and its impact on cloud applications' QoS". In: *IEEE Transactions on Cloud Computing* 3.4 (2015), pp. 449–458.

[16] Carlos Carvalho et al. "Avoiding Data Traffic on Smart Grid Communication System". In: (2014).

[17] Marcus Carvalho, Daniel A Menascé, and Francisco Brasileiro. "Capacity planning for IaaS cloud providers offering multiple service classes". In: *Future Generation Computer Systems* 77 (2017), pp. 97–111.

[18] *Cesium*. URL: http://cesium-ml.org/.

[19] Katja Cetinski and Matjaz B Juric. "AME-WPC: advanced model for efficient workload prediction in the cloud". In: *Journal of Network and Computer Applications* 55 (2015), pp. 191–201.

[20] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Computers & Electrical Engineering* 40.1 (2014), pp. 16–28.

[21] Zhijia Chen et al. "Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network". In: *Computational intelligence and neuroscience* 2015 (2015), p. 17.

[22] Jui-Sheng Chou and Thi-Kha Nguyen. "Forward Forecast of Stock Price Using Sliding-window Metaheuristic-optimized Machine Learning Regression". In: *IEEE Transactions on Industrial Informatics* (2018).

[23] Maximilian Christ et al. "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh–A Python package)". In: *Neurocomputing* (2018).

[24] Z. C. Dagdia et al. "A distributed rough set theory based algorithm for an efficient big data pre-processing under the spark framework". In: *2017 IEEE International Conference on Big Data (Big Data)*. Dec. 2017, pp. 911–916.

[25] Bruno L Dalmazo, João P Vilela, and Marilia Curado. "Online traffic prediction in the cloud: a dynamic window approach". In: *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE. 2014, pp. 9–14.

[26] Nickolas Allen Davis et al. "FailureSim: A System for Predicting Hardware Failures in Cloud Data Centers Using Neural Networks". In: *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE. 2017, pp. 544–551.

[27] Mahmood Deypir, Mohammad Hadi Sadreddini, and Sattar Hashemi. "Towards a variable size sliding window model for frequent itemset mining over data streams". In: *Computers & industrial engineering* 63.1 (2012), pp. 161–172.

[28] Sheng Di and Franck Cappello. "Fast error-bounded lossy HPC data compression with SZ". In: *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE. 2016, pp. 730–739.

[29] Ksenzovets Dmytro, Telenyk Sergii, and Pysarenko Andiy. "ARIMA forecast models for scheduling usage of resources in IT-infrastructure". In: *Computer Sciences and Information Technologies (CSIT), 2017 12th International Scientific and Technical Conference on*. Vol. 1. IEEE. 2017, pp. 356–360.

[30] Andi Drebes et al. "NUMA-aware scheduling and memory allocation for dataflow task-parallel applications". In: *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM. 2016, p. 44.

[31] Andi Drebes et al. "Scalable task parallelism for NUMA: A uniform abstraction for coordinated scheduling and memory management". In: *Parallel Architecture and Compilation Techniques (PACT), 2016 International Conference on*. IEEE. 2016, pp. 125–137.

[32] Harris Drucker et al. "Support Vector Regression Machines". In: *NIPS*. MIT Press, 1996, pp. 155–161.

[33] Hancong Duan et al. "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems". In: *Future Generation Computer Systems* 74 (2017), pp. 142–150.

[34] Lide Duan, Dongyuan Zhan, and Justin Hohnerlein. "Optimizing cloud data center energy efficiency via dynamic prediction of cpu idle intervals". In: *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE. 2015, pp. 985–988.

[35] Martin Duggan et al. "Predicting host CPU utilization in cloud computing using recurrent neural networks". In: *Internet Technology and Secured Transactions (ICITST), 2017 12th International Conference for*. IEEE. 2017, pp. 67–72.

[36] Thuan Hong Duong-Ba et al. "A Dynamic virtual machine placement and migration scheme for data centers". In: *IEEE Transactions on Services Computing* (2018).

[37] A. Egri et al. "Cross-correlation based clustering and dimension reduction of multivariate time series". In: *2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES)*. Oct. 2017, pp. 000241–000246.

[38] Timothée Ewart et al. "Performance Evaluation of the IBM POWER8 Architecture to Support Computational Neuroscientific Application Using Morphologically Detailed Neurons". In: *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. PMBS '15. Austin, Texas: ACM, 2015, 1:1–1:11. ISBN: 978-1-4503-4009-0. DOI: 10.1145/2832087.2832088.

[39] Wei Fang et al. "RPPS: a novel resource prediction and provisioning scheme in cloud data center". In: *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE. 2012, pp. 609–616.

[40] *Feature Selector*. URL: https://github.com/WillKoehrsen/feature-selector.

[41] Ronald Aylmer Fisher. *Statistical methods for research workers*. Genesis Publishing Pvt Ltd, 1925.

[42] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29 (2000), pp. 1189–1232.

[43] Keke Gai et al. "Resource management in sustainable cyber-physical systems using heterogeneous cloud computing". In: *IEEE Transactions on Sustainable Computing* 3.2 (2018), pp. 60–72.

[44] G. Gawde and J. Pawar. "Shape based time series reduction using PCA". In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. Mar. 2017, pp. 1–4.

[45] Li Ge and Li-Juan Ge. "Feature extraction of time series classification based on multi-method integration". In: *Optik-International Journal for Light and Electron Optics* 127.23 (2016), pp. 11070–11074.

[46] Mostafa Ghobaei-Arani, Sam Jabbehdari, and Mohammad Ali Pourmina. "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach". In: *Future Generation Computer Systems* 78 (2018), pp. 191–210.

[47] Ken Goodhope et al. "Building LinkedIn's Real-time Activity Data Pipeline." In: *IEEE Data Eng. Bull.* 35.2 (2012), pp. 33–45.

[48] *Google Cluster log.* URL: https://github.com/google/cluster-data.

[49] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. "Correlation and variable importance in random forests". In: *Statistics and Computing* 27.3 (2017), pp. 659–678.

[50] Jie Gu and Xiaomin Jin. "A simple approximation for dynamic time warping search in large time series database". In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2006, pp. 841–848.

[51] Tin Kam Ho. "Random Decision Forests". In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*. ICDAR '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 278–. ISBN: 0-8186-7128-9. URL: http://dl.acm.org/citation.cfm?id=844379.844681.

[52] Waheed Iqbal et al. "Adaptive Resource Provisioning for Read Intensive Multi-tier Applications in the Cloud". In: *Future Generation Computer Systems* 27.6 (June 2011), pp. 871–879.

[53] Jeremy Iverson, Chandrika Kamath, and George Karypis. "Fast and effective lossy compression algorithms for scientific datasets". In: *European Conference on Parallel Processing*. Springer. 2012, pp. 843–856.

[54] Yexi Jiang et al. "Cloud analytics for capacity planning and instant vm provisioning". In: *IEEE Transactions on Network and Service Management* 10.3 (2013), pp. 312–325.

[55] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.

[56] Murat Salim Karabinaoğlu and Tuba Gözel. "Load forecasting modelling of data centers and IT systems by using artificial neural networks". In: *Electrical and Electronics Engineering (ELECO), 2017 10th International Conference on*. IEEE. 2017, pp. 62–66.

[57] Samuel Karlin. *A first course in stochastic processes*. Academic press, 2014.

[58] In Kee Kim et al. "CloudInsight: Utilizing a Council of Experts to Predict Future Cloud Application Workloads". In: *IEEE International Conference on Cloud Computing*. 2018.

[59] Minyoung Kim. "Time-series dimensionality reduction via Granger causality". In: *IEEE Signal Processing Letters* 19.10 (2012), pp. 611–614.

[60] Diedirik P. Kingma and Jimmy L. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2015).

[61] *Kolmogorov-Smirnov test*. URL: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/ks.test.html.

[62] Jay Kreps, Neha Narkhede, Jun Rao, et al. "Kafka: A distributed messaging system for log processing". In: *Proceedings of the NetDB*. 2011, pp. 1–7.

[63] Mahendra Kutare et al. "Monalytics: online monitoring and analytics for managing large scale data centers". In: *Proceedings of the 7th international conference on Autonomic computing*. ACM. 2010, pp. 141–150.

[64] Christoph Lameter. "Numa (non-uniform memory access): An overview". In: *Queue* 11.7 (2013), p. 40.

[65] Richard P LaRowe Jr, Carla Schlatter Ellis, and Laurence S Kaplan. "The robustness of NUMA memory management". In: *ACM SIGOPS Operating Systems Review*. Vol. 25. 5. ACM. 1991, pp. 137–151.

[66] Charles L Lawson and Richard J Hanson. *Solving least squares problems*. Siam, 1995.

[67] Eun Kyung Lee, Hariharasudhan Viswanathan, and Dario Pompili. "Proactive thermal-aware resource management in virtualized HPC cloud datacenters". In: *IEEE Transactions on Cloud Computing* 5.2 (2017), pp. 234–248.

[68] Junnan Li et al. "SERAC3: Smart and economical resource allocation for big data clusters in community clouds". In: *Future Generation Computer Systems* 85 (2018), pp. 210–221.

[69] Min Li et al. "SparkBench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark". In: *Proceedings of the 12th ACM International Conference on Computing Frontiers*. CF '15. Ischia, Italy: ACM, 2015, 53:1–53:8. ISBN: 978-1-4503-3358-0. DOI: 10.1145/2742854.2747283. URL: http://doi.acm.org/10.1145/2742854.2747283.

[70] Shasha Liao et al. "Adaptive Resource Prediction in the Cloud Using Linear Stacking Model". In: *Advanced Cloud and Big Data (CBD), 2017 Fifth International Conference on*. IEEE. 2017, pp. 33–38.

[71] *linear trend feature*. URL: https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature_extraction.html#tsfresh.feature_extraction.feature_calculators.agg_linear_trend.

[72] David Littau and Daniel Boley. "Streaming data reduction using low-memory factored representations". In: *Information Sciences* 176.14 (2006), pp. 2016–2041.

[73] Chunhong Liu et al. "An adaptive prediction approach based on workload pattern discrimination in the cloud". In: *Journal of Network and Computer Applications* 80 (2017), pp. 35–44. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.12.017. URL: http://www.sciencedirect.com/science/article/pii/S1084804516303198.

[74] Liya Liu, Osman Hasan, and Sofiene Tahar. "Formal reasoning about finite-state discrete-time markov chains in HOL". In: *Journal of Computer Science and Technology* 28.2 (2013), pp. 217–231.

[75] Zoltan Majo and Thomas R Gross. "Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead". In: *ACM SIGPLAN Notices*. Vol. 46. 11. ACM. 2011, pp. 11–20.

[76] Karl Mason et al. “Predicting host CPU utilization in the cloud using evolutionary neural networks”. In: *Future Generation Computer Systems* 86 (2018), pp. 162–173.

[77] Llew Mason et al. “Boosting Algorithms as Gradient Descent”. In: *In Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 512–518.

[78] Matthew L Massie, Brent N Chun, and David E Culler. “The ganglia distributed monitoring system: design, implementation, and experience”. In: *Parallel Computing* 30.7 (2004), pp. 817–840.

[79] *Materna Trace*. URL: http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna.

[80] Marwa F Mohamed et al. “Data reduction in a cloud-based AMI framework with service-replication”. In: *Computers & Electrical Engineering* (2018).

[81] Meinard Müller. *Information retrieval for music and motion*. Vol. 2. Springer, 2007.

[82] Smrithy Girijakumari Sreekantan Nair and Ramadoss Balakrishnan. “Mitigating false alarms using accumulator rule and dynamic sliding window in wireless body area”. In: *CSI Transactions on ICT* (2018), pp. 1–6.

[83] Trung Hieu Nguyen, Mario Di Francesco, and Antti Yla-Jaaski. “Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers”. In: *IEEE Transactions on Services Computing* (2017).

[84] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. “Towards an autonomic auto-scaling prediction system for cloud resource provisioning”. In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2015, pp. 35–45.

[85] Mohd Halim Mohd Noor et al. “Adaptive sliding window segmentation for physical activity recognition using a single tri-axial accelerometer”. In: *Pervasive and Mobile Computing* 38 (2017), pp. 41–59.

[86] Tim Oates, Laura Firoiu, and Paul R Cohen. “Clustering time series with hidden markov models and dynamic time warping”. In: *Proceedings of the IJCAI-99 workshop on neural, symbolic and reinforcement learning methods for sequence learning*. Citeseer. 1999, pp. 17–21.

[87] *One and two-tailed tests*. URL: https://en.wikipedia.org/wiki/One-_and_two-tailed_tests.

[88] Tinghui Ouyang et al. “Model of selecting prediction window in ramps forecasting”. In: *Renewable Energy* 108 (2017), pp. 98–107.

[89] Tinghui Ouyang et al. “Optimisation of time window size for wind power ramps prediction”. In: *IET Renewable Power Generation* 11.8 (2016), pp. 1270–1277.

[90] Ashkan Paya and Dan C Marinescu. “Energy-aware load balancing and application scaling for the cloud ecosystem”. In: *IEEE Transactions on Cloud Computing* 5.1 (2017), pp. 15–27.

[91] Xuesong Peng and Barbara Pernici. “Correlation-model-based reduction of monitoring data in data centers”. In: *Smart Cities and Green ICT Systems (SMARTGREENS), 2016 5th International Conference on*. IEEE. 2016, pp. 1–11.

[92] Juan J Pérez-Solano and Santiago Felici-Castell. “Adaptive time window linear regression algorithm for accurate time synchronization in wireless sensor networks”. In: *Ad Hoc Networks* 24 (2015), pp. 92–108.

[93] *Perfmon2*. URL: http://perfmon2.sourceforge.net/.

[94] Iraklis Psaroudakis et al. "Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores". In: *Proceedings of the VLDB Endowment* 10.2 (2016), pp. 37–48.

[95] Iraklis Psaroudakis et al. "Scaling up concurrent main-memory column-store scans: Towards adaptive numa-aware data and task placement". In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1442–1453.

[96] Feng Qiu, Bin Zhang, and Jun Guo. "A deep learning approach for VM workload prediction in the cloud". In: *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE. 2016, pp. 319–324.

[97] Ali Asghar Rahmanian, Mostafa Ghobaei-Arani, and Sajjad Tofighy. "A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment". In: *Future Generation Computer Systems* 79 (2018), pp. 54–71.

[98] Thanawin Rakthanmanon et al. "Searching and mining trillions of time series subsequences under dynamic time warping". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 262–270.

[99] Célia G Ralha et al. "Multiagent system for dynamic resource provisioning in cloud computing platforms". In: *Future Generation Computer Systems* (2018).

[100] Oleksandr Rolik et al. "Dynamie management of data center resources using reinforcement learning". In: *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*. IEEE. 2018, pp. 237–244.

[101] Gordon J Ross et al. "Parametric and nonparametric sequential change detection in R: The cpm package". In: *Journal of Statistical Software* 66.3 (2015), pp. 1–20.

[102] Khalid Sayood. *Introduction to data compression*. Elsevier, 2005.

[103] Pavel Senin. "Dynamic time warping algorithm review". In: *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855 (2008), pp. 1–23.

[104] Filippo Seracini et al. "A Proactive Customer-Aware Resource Allocation Approach for Data Centers". In: *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*. IEEE. 2014, pp. 26–33.

[105] Prateek Sharma et al. "Managing risk in a derivative IaaS cloud". In: *IEEE Transactions on Parallel and Distributed Systems* 29.8 (2018), pp. 1750–1765.

[106] Balaram Sinharoy et al. "IBM POWER8 processor core microarchitecture". In: *IBM Journal of Research and Development* 59.1 (2015), pp. 2–1.

[107] GS Smrithy, Ramadoss Balakrishnan, and Nikita Sivakumar. "Anomaly detection using dynamic sliding window in wireless body area networks". In: *Data Science and Big Data Analytics*. Springer, 2019, pp. 99–108.

[108] Binbin Song et al. "Host load prediction with long short-term memory in cloud computing". In: *The Journal of Supercomputing* (2017), pp. 1–15.

[109] *Spark-Bench*. URL: https://github.com/SparkTC/spark-bench.

[110] Josep Subirats and Jordi Guitart. "Assessing and forecasting energy efficiency on Cloud computing platforms". In: *Future Generation Computer Systems* 45 (2015), pp. 70–94.

[111] T. Sun, H. Sun, and W. Chen. "Dimensionality reduction for Interval Time Series". In: *2012 World Congress on Information and Communication Technologies*. Oct. 2012, pp. 1115–1120.

[112] Ling Tang and Hao Chen. "Joint pricing and capacity planning in the iaas cloud market". In: *IEEE Transactions on Cloud Computing* 5.1 (2017), pp. 57–70.

[113] Dan Terpstra et al. "Collecting performance data with PAPI-C". In: *Tools for High Performance Computing 2009*. Springer, 2010, pp. 157–173.

[114] Sharda Tripathi and Swades De. "An Efficient Data Characterization and Reduction Scheme for Smart Metering Infrastructure," in: *IEEE Transactions on Industrial Informatics (2018)* (2018).

[115] Kishor S Trivedi and Andrea Bobbio. *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, 2017.

[116] Katharina Tschumitschew and Frank Klawonn. "Effects of drift and noise on the optimal sliding window size for data stream regression models". In: *Communications in Statistics-Theory and Methods* 46.10 (2017), pp. 5109–5132.

[117] Fan-Hsun Tseng et al. "Dynamic Resource Prediction and Allocation for Cloud Data Center Using the Multiobjective Genetic Algorithm". In: *IEEE Systems Journal* 12.2 (2018), pp. 1688–1699.

[118] *Tsfresh Features*. URL: https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html.

[119] Carlos Vazquez, Ram Krishnan, and Eugene John. "Time Series Forecasting of Cloud Data Center Workloads for Dynamic Resource Provisioning." In: *JoWUA* 6.3 (2015), pp. 87–110.

[120] Neal Wagner and Zbigniew Michalewicz. "An analysis of adaptive windowing for time series forecasting in dynamic environments: further tests of the DyFor GP model". In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM. 2008, pp. 1657–1664.

[121] Jianpei Wang et al. "An efficient data reduction method and its application to cluster analysis". In: *Neurocomputing* 238 (2017), pp. 234–244.

[122] Yongjian Wang and Hongguang Li. "A novel intelligent modeling framework integrating convolutional neural network with an adaptive time-series window and its application to industrial process operational optimization". In: *Chemometrics and Intelligent Laboratory Systems* (2018).

[123] Wei Wei et al. "Imperfect information dynamic stackelberg game based resource allocation using hidden Markov for cloud computing". In: *IEEE Transactions on Services Computing* 11.1 (2018), pp. 78–89.

[124] Joseph Nathanael Witanto, Hyotaek Lim, and Mohammed Atiquzzaman. "Adaptive selection of dynamic VM consolidation algorithm using neural network for cloud resource management". In: *Future Generation Computer Systems* 87 (2018), pp. 35–42.

[125] Kesheng Wu et al. "Statistical data reduction for streaming data". In: *Scientific Data Summit (NYSDS), 2017 New York*. IEEE. 2017, pp. 1–6.

[126] Ye Xia et al. "Large-scale VM placement with disk anti-colocation constraints using hierarchical decomposition and mixed integer programming". In: *IEEE Transactions on Parallel and Distributed Systems* 28.5 (2017), pp. 1361–1374.

[127] Ji Xue et al. "Spatial-Temporal Prediction Models for Active Ticket Managing in Data Centers". In: *IEEE Transactions on Network and Service Management* 15.1 (2018), pp. 39–52.

[128] Jun Yan et al. "Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing". In: *IEEE transactions on Knowledge and Data Engineering* 18.3 (2006), pp. 320–333.

[129] Jingqi Yang et al. "A cost-aware auto-scaling approach using the workload prediction in service clouds". In: *Information Systems Frontiers* 16.1 (2014), pp. 7–18.

[130] Tianqi Yu, Xianbin Wang, and Abdallah Shami. "A Novel Fog Computing Enabled Temporal Data Reduction Scheme in IoT Systems". In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–5.

[131] Matei Zaharia et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, pp. 2–2.

[132] Matei Zaharia et al. "Spark: cluster computing with working sets." In: ().

[133] Lei Zhan et al. "A Framework for Monitoring and Measuring a Large-scale Distributed System in Real Time". In: *Proceedings of the 5th ACM Workshop on HotPlanet*. HotPlanet '13. Hong Kong, China: ACM, 2013, pp. 21–26. ISBN: 978-1-4503-2177-8.

[134] Kaiyuan Zhang, Rong Chen, and Haibo Chen. "NUMA-aware graph-structured analytics". In: *ACM SIGPLAN Notices*. Vol. 50. 8. ACM. 2015, pp. 183–193.

[135] Qingchen Zhang et al. "An efficient deep learning model to predict cloud workload for industry informatics". In: *IEEE Transactions on Industrial Informatics* (2018).

[136] Weishan Zhang et al. "Resource requests prediction in the cloud computing environment with a deep belief network". In: *Software: Practice and Experience* 47.3 (2017), pp. 473–488.