國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

於多媒體霧計算平臺中預測資源可用性
Predicting Resource Availability in a Multimedia Fog Computing Platform

黃宜瑩
Yi-Ying Huang

學號：103062571
Student ID:103062571

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 105 年 10 月
October, 2016

國立清華大學
資訊工程研究所

碩士論文

於多媒體霧計算平臺中預測資源可用性

黃宜瑩 撰

105
10

中文摘要

隨著科技的進步，個人裝置（例如筆記型電腦與智慧型手機）擁有較以往更佳的硬體效能。 同時，各種不同的多媒體應用對於運算資源產生了逐漸增加的需求。 對於這樣的情境，我們採用多媒體霧計算（fog computing）平臺的概念，目標為減少使用雲端計算平臺的成本。 在此平臺中，服務提供者（fog provider）接收來自於服務使用者（fog users）的工作，並分配給工作者／裝置（fog workers/devices）。 對於實現此概念，主要有三個研究議題：（一）預測完成工作所需的資源、（二）預測裝置可提供的資源、（三）對於工作與可用資源進行排程。 本論文主要針對預測裝置可提供的資源進行研究。 我們採用三個機器學習演算法：隨機森林（Random Forest）、梯度提升樹（Gradient Boosting Tree）與神經網路（Neural Network），並使用開源函式庫實作。 我們使用兩組資料：使用者資料（desktop dataset）與數據中心資料（datacenter dataset），兩者的資源紀錄分別來自於真實使用者與數據中心裡的機器。 我們使用兩組資料的4/5進行10次交叉驗證，得出機器學習演算法所需的超參數（hyperparameter）。 結果顯示兩組資料所需的最佳超參數是不同的。 從此可知，當服務提供者採用新的資料，或是資料有大量變異時，必須重新微調超參數。 我們使用兩組資料剩下的1/5以及真實的動畫處理資料來進行模擬。 實驗結果顯示：（一）神經網路演算法對於兩組資料可達到預測值與實際值分別僅6.08%與2.00%的差異、（二）較準確的可用資源量預測可使失敗的工作量減少。

Abstract

The personal devices such as laptops and smartphones are being equipped with better hardware, which leads to stronger computing abilities. At the same time, the demand of various multimedia applications requires increasing computational resources. We propose to build the multimedia fog computing platform, which aims at reducing the cost of using cloud computing. In this platform, the fog provider receives the jobs from the fog users and schedules them to the fog workers/devices. There are three main research problems: (i) prediction of the required amount of resources of the jobs, (ii) prediction of the available resources of the fog devices, and (iii) scheduling the jobs and the fog devices. This thesis focuses on the prediction of the amount of the available resources. We adopt three machine learning algorithms, namely, the Random Forest, Gradient Boosting Tree, and Neural Network, and implement them using open source libraries. We apply two datasets, desktop and datacenter datasets, where the traces come from real users and machines in the cloud datacenter, respectively. We use 80% of both datasets and perform 10-fold cross validation to fine-tune the hyperparameters of the proposed algorithms. The optimal combinations of the hyperparameters for both datasets are different. We learned that when the fog provider applies new datasets, or when the dataset dramatically changes, it is necessary to re-tune the hyperparameters. We implement a simulator and use the rest 20% of both available resource datasets and a real animation rendering jobs dataset to drive our simulator. The simulation results show that: (i) the Neural Network-based algorithm achieves 6.08% and 2.00% deviation in average for the desktop and datacenter datasets, respectively, and (ii) more accurate prediction of the amount of available resources leads to fewer failed jobs.

Contents

TSINGHUA UNIVERSITY

List of Figures

List of Tables

Chapter 1

Introduction

While the technologies are advancing, personal devices such as laptops and smartphones are being equipped with better hardware, which leads to stronger computing abilities. Meanwhile, the demand of various multimedia applications requires increasing computational resources. To meet the demand, one possible solution is to rent the cloud servers. However, it is expensive to fulfill extreme demands of computational resources solely by purchasing cloud services. There are several limitations of the cloud computing. For example, the cloud datacenter is far away from the users, which leads to long response time. We propose to build a platform for the multimedia applications based on the concept of *fog computing*, which aims at reducing the cost of using cloud computing. We integrate the resources from the cloud, the edge cloud, and the fog in our platform. The cloud contains powerful machines. The edge cloud consists of devices such as routers and WiFi access points. The fog contains personal devices such as desktop computers and smartphones. There are many advantages by doing so. For example, the fog provides many kinds of resources, including computational, communicational, storage, and sensory resources. Moreover, it reduces the network traffic by not sending everything to the cloud.

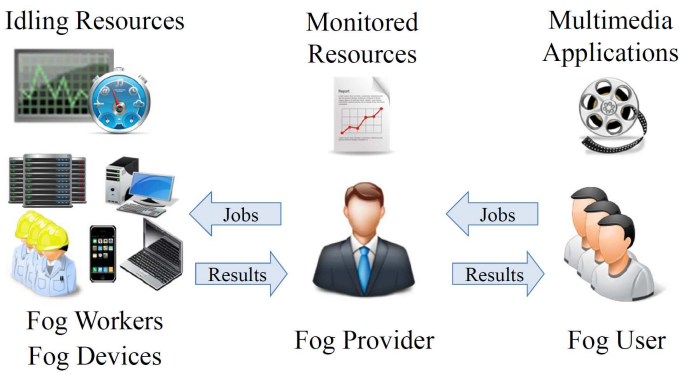


Figure 1.1: Overview of our multimedia fog computing platform.

Fig. 1.1 shows the overview of our fog computing platform for multimedia applications. The platform, or *fog provider*, is the main manager of the system. It harvests the computational, communicational, and storage resources from the *fog devices*, and then offers these resources to the *fog users*. The fog devices include computing machines such as the desktop computers and laptops, as well as the smart devices such as smartphones and tablets. These devices are characterized by their dynamic workload, i.e., they may not always be fully loaded. For example, a desktop only working on simple document processing or a smartphone being charged may have much idling resources for extra computing tasks. The fog provider purchases these otherwise wasted resources from the devices owners, called *fog workers*, and resell these idling resources to the fog users. Compared to the cloud computing platform, the fog computing platform is much cost-effective by saving costs of (i) buying factories and machines for the datacenter establishments, (ii) building power distribution, cooling, and networking systems, (iii) paying the electricity consumption, and (iv) hiring administrative staffs to maintain the datacenter.

We consider the animation rendering as the sample multimedia fog computing application. Animation rendering consumes significant computing resources, and the resource demand is steadily increasing. In 1995, Toy Story required 800,000 machine hours to render at 2 to 15 hours per frame [4]. In 2001, when rendering the Monsters, Inc., Pixar spent about 12 hours to render a single frame with the main character in it [5]. In 2014, Disney even needed to render Big Hero 6 on a 55,000-core supercomputer farm [2]. As the quality of animation film improves, the demand and the cost on the rendering farm increase as well. It is a considerable cost to build a rendering farm or renting cloud resources to render the animation films. Therefore, we use animation rendering as a candidate application for the multimedia fog computing platform.

While fog computing has aforementioned advantages for animation rendering, the fog devices and fog workers are characterized by their heterogeneity and uncertainty. Fog devices include various computing devices with different specifications. The different hardware integration leads to different computational capabilities. One animation rendering job may take different time periods to finish, which depends on the computational power of the devices. In addition, the amount of residual resources may also effect the job completion time, e.g., a fully loaded desktop computer with better hardware equipments may not perform better than a mostly idling laptop. Even if a fog device behaves constantly, its owner, the fog worker, may act uncertainly. For example, the fog worker may turn on/off the device anytime, forget to charge up the device, or move to a region where the network signal is too weak to connect to the Internet.

There are three main research problems for our multimedia fog computing platform. The first one is to analyze the jobs from the fog users. Since there are various multi-

media applications, it is challenging to know the exact execution time and required type and amount of resources. In our scenario, we consider the animation rendering as the application. It requires a large amount of computational resources since nowadays, each frame contains so many materials. It also requires large enough storage space to store the raw data and the rendered results. If the fog user sets an urgent deadline for retrieving the results, then stable and enough communicational resource is also necessary. Hence, to better utilize the resources, it is important to analyze the jobs. The second one is to discover the usage pattern of the fog workers/devices and predict the resource availability. We assume that each fog worker has his own habit of using his device. It may be effect by personal lifestyle or his occupation. Whether and how much the usage pattern is predictable differ from person to person. If the fog provider is able to accurately predict the available resources, he will be able to know how much resources can be used in advance, which may greatly help the job scheduling. After the fog provider is able to tackle the previous two problems, the last problem is to schedule the jobs and the available resources. To improve the overall performance of our platform, there are several other issues to be solved. For example, the predicted results may turn out to be wrong. In case the assigned job cannot be completed as expected, the fog provider may need to decide the number of redundant rendering jobs. Namely, one rendering job is assigned to multiple fog devices to guarantee that at least one of them will be finished and submitted back successfully. However, this consumes additional resources. The tradeoff of increasing the redundancy and avoiding resource wastes is a critical problem. The previous work of our lab [23] has studied the first research problem. As an extension of this research project, this thesis focuses on predicting the available resources of the fog devices.

Chapter 2

Related Work

2.1 Fog Computing

The concept of *fog computing* was first proposed for Internet-of-Things applications [14]. Since the centralized data centers are unable to support the latency-sensitive applications, Bonomi et al. proposed the concept of fog computing. They defined the fog computing as the extension of cloud computing to the edge networks. The characteristics of the very large amount of nodes include the mobility, heterogeneity, and wide-spread geographical distribution. Hence there are several challenges such as the orchestration and management of the fog nodes. The Vaquero et al. [32] proposed a generalized definition of the fog computing. They further included the resources of the end devices owned by the public crowds into the definition. Collectively leveraging the resources from datacenter, edge networks, and the end devices, the generalized fog computing provides better support for multimedia applications. Since it takes advantages of heterogeneous computing resources, fog computing is a feasible choice for the delay-sensitive and resource-hungry multimedia applications. The authors also pointed out more open challenges ahead. Due to the significant heterogeneity, it is critical to take care of the problems such as discovery and synchronization applications among the fog nodes as well as the hardware limitations of the devices.

There are some previous works similar to our concept of the multimedia fog computing platform. Sarmenta proposed the concept of volunteer computing [30], which aggregately utilizes the idling resources of desktops to perform computationally-demanding jobs, which is similar to our concept. The author mentioned several research issues including accessibility, applicability, reliability, and economic issues. However, our platform is different from the volunteer computing for several aspects. For example, volunteer computing utilizes the resources from the volunteers. Therefore, they do not need to carefully manage the resources. SETI@Home [12] is an experiment in public resource comput-

ing. It is a volunteer computing application, which analyzes radio signals from the space and aims at detecting intelligent life outside the earth. This project was launched in May 1999. [7] There have been millions of participants involved in this project, which is a very large-scale volunteer computing project. BOINC [11] is a generalized platform of volunteer computing. It collectively aggregates the idling resources of the participants and utilizes them for computational jobs of various fields such as mathematics, medical science, environmental science, and astrophysics. Participants installed their application can start contributing resources right away after connecting to the Internet. Different from our multimedia fog computing platform, BOINC attracted the volunteers by diverse long-term and high-profile projects, such as finding aliens. Hence, this platform does not face some of our challenges, e.g., fog providers must guarantee that a job be completed on time in order to meet the demands of the fog users.

2.2 System Modeling

Several studies have worked on the availability modeling and system modeling in the literature. Javadi et al. [24] aimed at discovering the subsets with similar characteristics and the availability models in a distributed system. The authors applied three randomness tests and finds out 21% of hosts from the SETI@home dataset whose availability is independent of others and is distributed in the same way. Then they applied k-means and hierarchical clustering on those hosts, which optimally results in six clusters. The models are evaluated using a resource brokering problem. The results showed that their proposed models are helpful for scheduling problem. Kondo et al. [26] also studied the availability patterns and performed clustering techniques on the hosts. The authors examined hour-in-day and day-of-week time features. After detecting the patterns, they used k-means clustering to determine hosts whose availability exhibits similar time effects, which results in five clusters. The authors used a barrier synchronization application to show how their correlated clusters improved the resource management in volunteer computing. Rood et al. [29] presented the diurnal pattern of the user behavior through trace analyses. The authors proposed a multi-state availability model with five different availability states and compared different predictors. They evaluated the accuracy of the predictors, which outperforms existing solutions. They then proposed their job replication technique which improved job makespan with little redundancy. Shang et al. [31] considered user behavior patterns on different days of a week. This study took advantages of Dempster's rules of combination. The simulation results showed that their model can take advantages of the reliable nodes, which reduces the communicational burden and improves the processing power. Dabrowski et al. [18] utilized Markov chain model to simulate large-scale grid

systems. The commercial success of the grid technology makes it necessary to develop an analytical tool for simulating a complex system. The authors first proposed the state transition model for one task and then aggregated multiple task states, which is represented as a piece-wise homogeneous Markov chain. Their model simulates a grid system, and the Markov chain procedure consumes little computational resources.

2.3 Availability Prediction

Andrzejak et al. [13] computed a Naïve Bayes classifier for each host to predict the CPU availability. The authors assumed that the CPU of a host is either 100% available or 0%. The evaluation results using traces from SETI@home project showed that the proposed solution can achieve 95% or greater accuracy. This study performed classification, which only predicts whether a host is available or not. Our predictor performs regression, which aims at a more accurate resource usage value. Brevik et al. [15] used one parametric and two non-parametric prediction techniques to predict how long a host will be available. The authors applied three different datasets and compared the performances of three techniques. The experimental results showed that the parameter estimate technique is sensitive to the characteristics of the applied data and the Binomial Method performs the best. Carvalho et al. [17] proposed a prediction model for resource availability of a peer in a peer-to-peer (P2P) desktop grid. The proposed method is based on the Network of Favours incentive mechanism. The authors found that higher resource contention leads to larger prediction errors. Ramachandran et al. [28] also studied the resource availability prediction in a P2P desktop grid. In their architecture, the monitoring and prediction engine logs the resource and group availability data periodically. They used the previous week's data to calculate the current week's availability. The prediction results help reduce the job migrations (job interruptions). Akioka et al. [10] applied Markov model-based meta-predictor for one-step-ahead prediction of the CPU and network load. Their solution takes seasonal variation for both resources into consideration. The proposed method can perform prediction for hours to days. To better schedule the resources in a grid environment, Wu et al. [33] proposed a hybrid model to predict the available grid resources. Their solution integrates autoregressive model and two filters, the Kalman filter [25] and Savitzky-Golay smoothing filter [27]. They first reduced the data noise by using the two filters, recursively computed autoregressive coefficients and predicted the value for future time points, and filtered the results for smoothing. The results showed that they can achieve up to 50-step-ahead prediction with prediction mean square error of 0.04 on average. Yuan et al. [35] improved this work by proposing a parameter-level adaptive method based on the previous hybrid model. Yang et al. [34] also proposed a

multi-step-ahead prediction approach. Instead of directly predict the load statistics, they decomposed the steps. They first predicted the change range of the CPU load, and then predicted the change direction, i.e., increase or decrease. Then they composed both results as the final prediction. When performing the composition, weighting strategies, including majority rule strategy and uniform decline strategy, and machine learning algorithm, Adaboost algorithm, are applied. Doulamis et al. [20] used 3-D rendering as the application. The authors considered both modeling and workload prediction using a combined fuzzy classification and neural network. They investigated three rendering algorithms, the ray tracing, the radiosity, and the Monte Carlo irradiance analysis. They conducted experiments which showed the great performance of their proposed solutions. Di et al. [19] designed a prediction method using Bayesian model and investigated the most effective combination out of a set of candidate features. They used Google trace which includes more than ten thousand heterogeneous hosts for experiments. They compared their proposed solution against several other algorithms, including methods related to the moving averages, auto-regression, and noise filters. They run several types of evaluation with different combinations of training and testing periods. The results show that their proposed solution outperforms others in terms of the long-term prediction accuracy. They also improved the precision of the pattern prediction under a load balancing scenario. Gmach et al. [22] characterized the workload demand patterns and proposed a workload placement service. They aimed at efficiently utilizing the resource pools for a large number of enterprise services. They run the evaluation using the data of six months involving more than one hundred enterprise applications. Their results showed that the prediction accuracy achieves resource savings. They also found that the workload trend prediction is helpful when it is used for reasonably recent trend.

Chapter 3

Research Problem

The fog provider needs to schedule jobs from the fog users and the available resources from the fog devices. However, it is impossible to know exactly how much resources the fog devices can provide in a certain future period. Thus we use historical data to predict this information. It is naive that accurate prediction of the available resources can help job scheduling in our platform. Each user may have his own usage pattern, which includes daily and weekly regularities. We apply machine learning predictors as our solutions to predict the resource availability in a future time period.

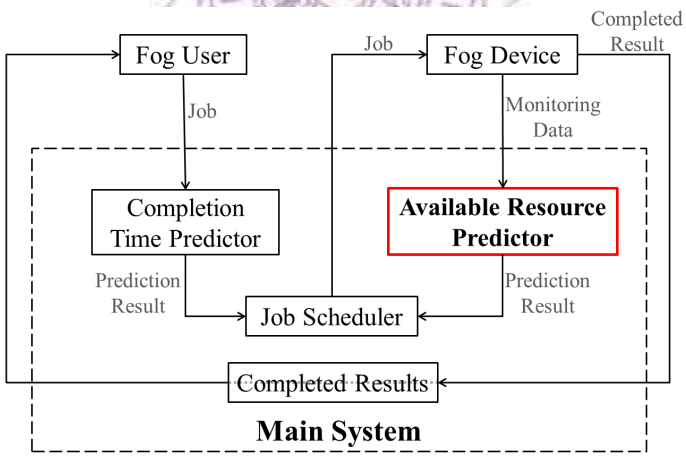


Figure 3.1: The architecture of our multimedia fog computing platform.

Fig. 3.1 illustrated the architecture of our multimedia fog computing platform. Our platform works as follows. When a person agrees to sell the idling resources of his machine, some general information about this fog devices will be checked, such as the number of its CPU cores, the CPU speed, and the amount of its RAM. Then the fog provider continuously monitors the resource usage of the fog workers. The monitored resources include (i) computational resources, such as CPU and RAM utilization, (ii) storage resources, such as disk utilization and its I/O speed, and (iii) communicational

resources, such as the network type, speed, and transmitted/received amount of data. Above-mentioned resources are sampled at a predefined frequency. The statistics along with the timestamp are stored as the historical data. The available resource predictor uses the historical resources usage data to predict the amount of available resources in a future period. We assume that a fog worker with steady usage pattern may be more easily to predict. For example, a student who keeps regular hours may usually charge his smartphone at night, which consequently has much idling resources during some certain hours of a day. The available resource predictor uses historical data to make prediction, and uses actual data to compare with the predicted results to see whether a fog worker has a steady usage pattern. On the other hand, the characteristics of the rendering jobs, such as the number of frames, polygons, and textures, are recorded as well. These data are used to predict the completion time of the rendering jobs. Then the job scheduler uses the predicted available resources and the completion time to schedule the job assignment to the fog devices/workers. The rendering jobs will be distributed to the fog devices according to the assignment from the job scheduler. After the fog devices finish rendering, the rendered results will be sent back to the fog users. The amount of consumed resources and the actual completion time are recorded and sent back to the fog provider. The actual statistics can be used to compare with the predicted ones, which can help the fog provider more accurately perform the predictions. We highlight the available resource predictor in fig. 3.1 since it is the main focus of this thesis.

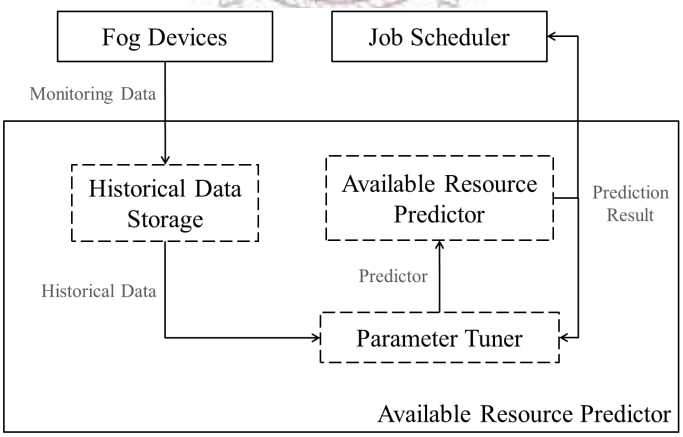


Figure 3.2: The architecture of the available resource predictor.

Fig. 3.2 gives a more detailed architecture of the available resource predictor. As we mentioned above, the general device information and monitored resource usage data will be stored in the historical data storage. Currently, we assign each fog device a serial number as the unique ID. To generate the predictor, we have a parameter tuner. It is responsible for tuning the hyperparameters and the parameters. Hyperparameters are the

parameters that are chosen by human beings offline, which can be tuned using the large historical data. By contrast, parameters are updated more frequently to deal with the dynamics of the data, e.g., new versions of fog applications, or online updates of the model parameters. With the fine-tuned parameters, we generate the predictor, which produces the prediction results and sends to the job scheduler on demand.

Then we describe our regression problem in detail. Let $\mathbf{F}$ be the set of all the machine learning algorithm predictors that the fog provider uses. Let $\mathbf{R}$ be the set of all kinds of resources that the fog provider takes into consideration when he is scheduling the jobs. For each algorithm $f \in \mathbf{F}$, we determine its hyperparameters and train the parameters for each of the dataset, which we denote by $\mathbf{H}_f$ and $\mathbf{P}_f$, respectively. Let $\mathbf{D}$ be the historical data of the resource availability. $\mathbf{D}$ is a matrix of the historical data, where each row consists of the amount of available resources and the collected features. For each considered resource $r \in \mathbf{R}$, $\mathbf{D_r}$ represents that each row consists of the amount of available resource r and the rest of the collected features. The objective of our regression problem is to obtain the estimated value $\widetilde{v}_{r,t}$ of the available resource r of a future time instant t, where $v_{r,t}$ is the actual value of the available resource r of that future time instant t. We aim at predicting the results of a future time period, i.e., a set of successive time instants. To simplify the formulation, successive $\widetilde{v}$ and v within a time period $\mathbf{T}$ are rearranged to the vector $\widetilde{y}_{r,\mathbf{T}}$ and $y_{r,\mathbf{T}}$, representing the estimated and actual values of the future time period $\mathbf{T}$, respectively. The regression predictor can be characterized as

$$\widetilde{y}_{r,\mathbf{T}} = \mathbf{F}(\mathbf{D_r}, \mathbf{H}, \mathbf{P}, \mathbf{T}), \tag{3.1}$$

where we call y the *target* of our regression problem. That is, the fog provider first chooses one of the machine learning algorithm predictors and the target resource r to predict. The historical data, the hyperparameters and the parameters for this predictor, and the target time interval are used as inputs. Then the predictor outputs the estimated resource availability. Note that the predictors for each considered resource are computed separately. It is because that we believe that for each fog worker, different resources should have its own pattern. For example, for a desktop owned by a student, the computational resource such as CPU may go high during daytime because he is working hard or playing games. Then the usage goes down during nighttime because the student goes to sleep. As for the communicational resource such network throughput, on the contrary, it may go high during nighttime because the student is downloading large files when he is sleeping. Considering the difference of the resource usage patterns, we choose to compute predictors for each resource separately.

Chapter 4

Solutions

We give details about our three solution approaches in this section. Since we conduct *data-driven* simulations, we describe the datasets we employed here. Then we show the process of tuning the hyperparameters of our predictors.

4.1 Solution Approaches

There is no standard answer for which machine learning algorithm best fits a certain prediction problem. Each algorithm may lead to its accuracy and execution time depending on how it constructs the predictor. The fog provider may choose the algorithm according to the demands, such as the characteristics of the input data, execution time, or the accuracy. We consider three state-of-the-art machine learning approaches, Random Forest, Gradient Boosting Tree, and Neural Network. Namely, we let $\mathbf{F} = \{\mathrm{RF}, \mathrm{GB}, \mathrm{NN}\}$. Three algorithms have been used for regression problems, such as Web search ranking, human pose estimation, computer vision, and speech handwriting recognition. Based on these three algorithms, we have developed Random Forest-based (RF-based), Gradient Boosting Tree-based (GB-based), and Neural Network-based algorithms using open source libraries from scikit-learn and xgboost [6, 8].

Both RF-based and GB-based algorithms are tree-based ensemble methods, but they are different from each other in the way of constructing the prediction model. Ensemble methods combine the prediction results of multiple base estimators to improve the robustness over one single estimator. Although using multiple estimators consumes longer computation time, it usually leads to better accuracy. Ensemble methods construct their base estimators with a given learning algorithm. RF-based algorithm generates a number of independent decision trees. Each decision tree is trained with random samples from the training data. That is, when RF-based algorithm constructs a decision tree, it randomly selects several features and then randomly specifies a threshold value for these

features. It splits the dataset using previously selected features and thresholds, and selects the new feature and the threshold that reduce the highest data entropy. RF-based algorithm then makes predictions by averaging all decision trees' prediction results. The bias of RF-based algorithm usually slightly increases due to the randomness existed in this algorithm. However, the averaging decreases the variance, which can compensate for the increase because of the bias. Hence it can perform better than a single decision tree. In contrast, GB-based algorithm consists of a sequence of trees in a stage-wise fashion. Each successive tree is built to predict the residuals of the preceding one. GB-based algorithm's trees are trained and combined using a more sophisticated weighting scheme. Readers are referred to Friedman et al. [21] for more details on Random Forest and Gradient Boosting Tree. NN-based algorithm is a relatively complicated method. The neural network consists of multiple layers, including input layer, hidden layer, and output layer. Each layer contains one or more corresponding neurons, i.e., the input/output layer has input/output neurons, and the hidden layer has hidden neurons. In the neural network, neurons at each layer are connected to the ones at the next layer. These connection are called synapses. Each synapse has its weight, which is calculated during the training process. At the beginning of the training phase, the neural network used the input neurons and random weights to generate the output neurons. The predicted value are compared with the actual output. Then the results are used to adjust the weights of the synapses. The weights is updated according to the learning rate, which is a parameter that can be determined for this solution.

4.2 Trace Collection & Used Datasets

To run the simulations, we build up a trace collection program using C++. We call this self-collected traces as *desktop dataset* for simplicity and denote it by $\mathbf{D}_{\text{de}}$. We use the open APIs provided by Microsoft [3] to get the machine information and resource usage statistics. Our program generates two files and stores them locally on the machine. Both recorded data of the computational, storage, and communicational resources. The file containing the machine information records the following information:

- **Computational resources:** the CPU speed, the number of the cores, the size of the virtual memory, and the size of the physical memory.
- **Storage resources:** the size of the disk.
- **Communicational resources:** the description string of the network interface in use and the type (Ethernet, IEEE 802.11 Wireless, IEEE 1394 Firewire, IEEE 802.16 WiMax, and others).

The other file containing the resource usage statistics records the following information:

- **Computational resources:** the CPU utilization, the used amount of the virtual memory, and the used amount of the physical memory.
- **Storage resources:** the used amount of the disk and the read/write speed of the disk.
- **Communicational resources:** the maximal speed of the network interface and the amount of the received/transmitted data.

The trace collection program records the above statistics with the timestamp every 10 seconds. We find 25 volunteers for the trace collection where most of them are graduate students. Volunteers install our program on their machine (desktop computers or laptops) and submit the generated files. Most of them say that the resource monitoring program did not make any effects to their daily use. Only one has the situation that the program uses so much memory that he needs to restart it from time to time.

We further employ a *performance metrics dataset* [1] to drive our simulator. The performance metrics dataset is the resource usage records of the virtual machines (VMs) of a datacenter from BitBrains, a service provider serving many enterprises. The datacenter dataset contains the resource usage statistics, including:

- **Computational resources:** the CPU cores, the CPU capacity, the CPU utilization, the size of memory, and the memory usage.
- **Storage resources:** the disk read/write throughput.
- **Communicational resources:** the network received/transmitted throughput.

The dataset contains two subsets. One records the statistics of $1,250$ VMs for 1 month, while the other one records that of 500 VMs for 3 months. We only adopt the latter one with statistics of 500 VMs, where the collection period is between July and September 2013. For simplicity, we call the adopted sub-dataset the *datacenter dataset* in the following manuscript and denote it by $\mathbf{D}_{\text{da}}$.

The above-mentioned two datasets are used in our simulations, i.e., $\mathbf{D} = \{\mathbf{D}_{\text{de}}, \mathbf{D}_{\text{da}}\}$. From the collected data, our considered resources are the utilization of CPU, the utilization of memory, the usage of disk, and the network throughput, i.e., $\mathbf{R} = \{\text{CPU}, \text{MEM}, \text{DISK}, \text{NWK}\}$. As we have mentioned in 3, we choose to compute predictors for each resource separately since we consider the difference of the resource usage patterns. In the rest of the thesis, we consider the utilization of CPU as an example. Namely, we let $r = \text{CPU}$ and use $\mathbf{D}_{\text{CPU}}$ as the training data, if not otherwise specified. For job scheduling, we temporarily

only consider the CPU resources for now. Taking other resources into consideration for scheduling is a part of our future work. We use two datasets due to the following reasons. Our platform aims at recruiting fog workers who are real users in the world. To get a more convincing dataset, we build up our own resource monitoring program and invite real users to install it. During the trace collection, we encounter the situations that may also happen to the fog provider, e.g., the fog devices are turned off or are disconnected from the Internet from time to time. We believe that the desktop dataset would produce the simulation results closer to the reality. Nevertheless, due to the limit of time and resources, we are unable to collect the statistics for such a long period and a large amount of users. Thus we employ the datacenter dataset. This dataset is the records of VMs in a datacenter, where the VMs are launched according to the users' requirements. Normally, there would not be idling resources. Since the enterprises pay for the resources they rent, they would not waste any of them. While we want to get insight into the idling resources of the machines, this case may not perfectly fit our scenario. Nevertheless, we can consider the required resources from the enterprises as the consumed resources by a real user. Then there may also be usage patterns in this dataset. The reality of the desktop dataset and the quantity of the datacenter dataset are the main reasons we adopt two datasets.

As listed above, we collected usage statistics including computational, storage, and communicational resources. According to different resource demands of the jobs from the fog users, the fog provider may need to predict different resources in a future period accordingly. In our simulation, we use the CPU utilization, i.e., the percentage of the CPU usage, as our prediction target. Since we record the general information of each fog device, we can acquire the exact amount of predicted available CPU resources by multiplying the CPU utilization by the CPU speed. When we finish prediction, we use the actual amount of predicted CPU resources to schedule the rendering jobs.

	Datacenter Dataset	**Desktop Dataset**
Type of node	VM in datacenter	desktop or laptop of a real user
Total # of nodes	500	25
Period	3 months	1 month
Sampling frequency	5 minutes	10 seconds
Total # of records	12,496,728	2,967,335
Avg. # of records	24,993	118,693
size of training set	9,997,696	2,373,909
size of testing set	2,499,032	593,426
# of features	9	9
Prediction target	CPU utilization (%)	CPU utilization (%)

Table 4.1: Sample statistics of datacenter dataset and desktop datasets.

Table 4.1 provides some sample statistics of datacenter and desktop datasets. As aforementioned, the datacenter dataset contains records of 500 VMs and the recording period lasts for 3 months. The sampling frequency is 5 minutes per record. There are $12,496,728$ records in total. Averagely, each VM has $24,993$ records.

According to Abu-Mostafa et al. [9], it is recommended to reserve a portion of the dataset for testing, where the practical rule of thumb is to reserve $1/5$ of the dataset. Following the suggestions, we reserve $1/5$ of the whole dataset as the testing set for the final testing. Namely, we perform the training procedure, i.e., the 10-fold cross validation, on the rest $4/5$ of the dataset as the training set. The testing set is used to simulate the future data and is completely exclusive to the training procedure. The reason is to prevent the generated predictor from knowing the future data in advance, which keeps the fairness and cleanness.

Therefore, for the datacenter dataset, we reserve 80% of the dataset as training set and the rest 20% as testing set. That is, $9,997,696$ and $2,499,032$ records are used for training and testing, respectively. On the other hand, the desktop dataset contains records of 25 real users' personal devices, including desktop computers and laptops, where the recording period lasts for 1 month. The sampling frequency is 10 seconds per record. There are $2,967,335$ records in total. Averagely, each node has $118,693$ records. We split this dataset in the same way. That is 80% (which equals to $2,373,909$) and 20% (which equals to $593,426$) of the dataset are used as training and testing set, respectively.

Both datasets have the following 9 features:

- id: a unique ID for each node as an integer, e.g., 1, 2, ...
- epoch: the number of seconds since 1970-01-01 08:00:00 as an integer, e.g., 1451577600 for 2016-01-01 00:00:00.
- dayInMonth: the day of the month as an integer, e.g., 1, 2, ..., 31.
- dayInWeek: the day of the week as an integer, where Monday is 1 and Sunday is 7, e.g., 1, 2, ..., 7.
- isWeekend: whether the epoch time is between Saturday and Sunday as an boolean value, e.g., 0 and 1.
- hourInWeek: the hour of the week as an integer, e.g., 0, 1, ..., 167.
- hourInDay: the hour of the day as an integer, e.g., 0, 1, ..., 23.
- minute: the minute of the hour as an integer, e.g., 0, 1, ..., 59.
- daySlot: which slot of the day as an integer, where a day is split into three slots, e.g., 0 for 0 a.m. to 7 a.m., 1 for 8 a.m. to 3 p.m., and 2 for 4 p.m. to 11 p.m.

The prediction target for the two datasets are both CPU utilization (%). Since we possess the CPU speed of each node, it is equivalent to predict the exact used CPU and the CPU utilization. To keep a smaller data scale, we choose to predict the CPU utilization. When the prediction results is used for scheduling, the scheduler multiplies the utilization by CPU speed to obtain exact CPU resources for decision making.

4.3 Optimal Hyperparameters

Hyperparameters are the preconfigured parameters that can not be learned during the training process. These parameters effect the structure of the prediction model. That is, different values of the hyperparameters lead to completely different models. They are chosen by human beings offline by applying historical data.

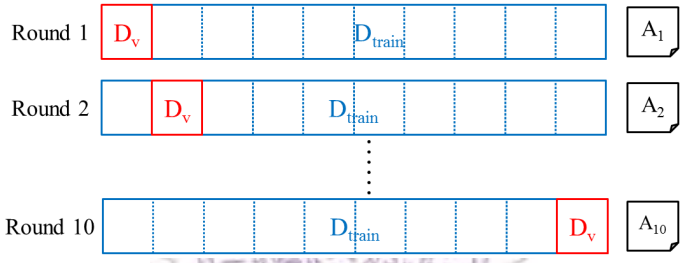


Figure 4.1: The procedure of 10-fold cross validation.

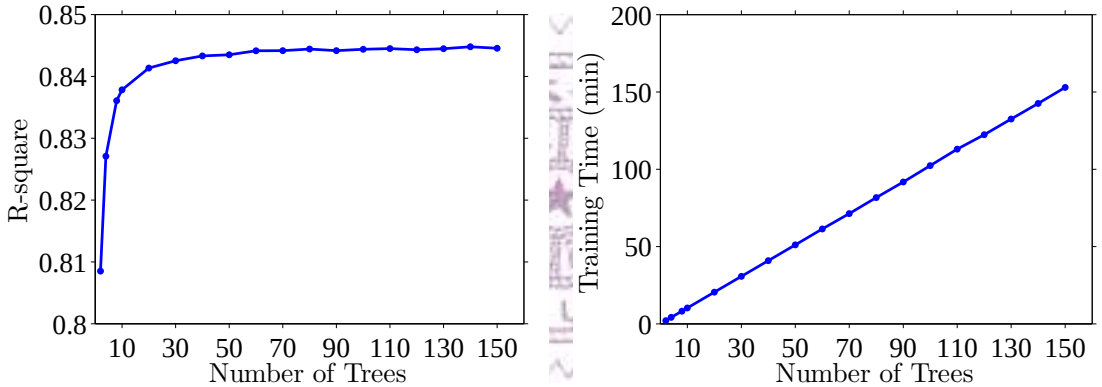
For training, we perform V-fold cross validation to acquire the optimal hyperparameters for the RF- and GB-based predictors. According to Abu-Mostafa et al. [9], V-fold cross validation is generally preferred over single validation. 5-fold and 10-fold are usually adopted. In practical, the rule of thumb is $V = 10$. Hence we perform 10-fold cross validation for training. Fig. 4.1 illustrates the procedure of the 10-fold cross validation. The whole dataset is split into 10 equal-sized folds. In each round, one fold is used as the validation set D_{v}, and the rest 9 folds are used as the training set D_{train}. The model is generated by applying the training set and then evaluated by the validation set. This procedure repeats 10 times until each of the 10 folds has been used as D_{v} once. Each round i generates an performance score A_i. The final performance is the average of the 10 performance scores: $A = \frac{1}{10}\sum_{i=1}^{10} A_i$. Fig. 4.2 illustrates the pseudocode of performing the k-fold cross validation.

There are several hyperparameters for RF- and GB-based predictors. We perform the cross validation on different combinations of values and choose the set with the best performance. Note that we do not train the hyperparameters of both algorithms for both datasets on the same machine. Because training the various combinations consumes much

1: Let $\mathbf{A}[1, 2, \ldots, k]$ be a new array
2: Divide the training data into k folds
3: **for** $i = 1$ to k **do**
4: Train and generate the model using $k - 1$ folds except the i^{th} fold
5: Test the model using the i^{th} fold and calculate the performance score a
6: $\mathbf{A}[i] = a$
7: **return** $Average(\mathbf{A})$

Figure 4.2: The pseudocode of performing the k-fold cross validation.

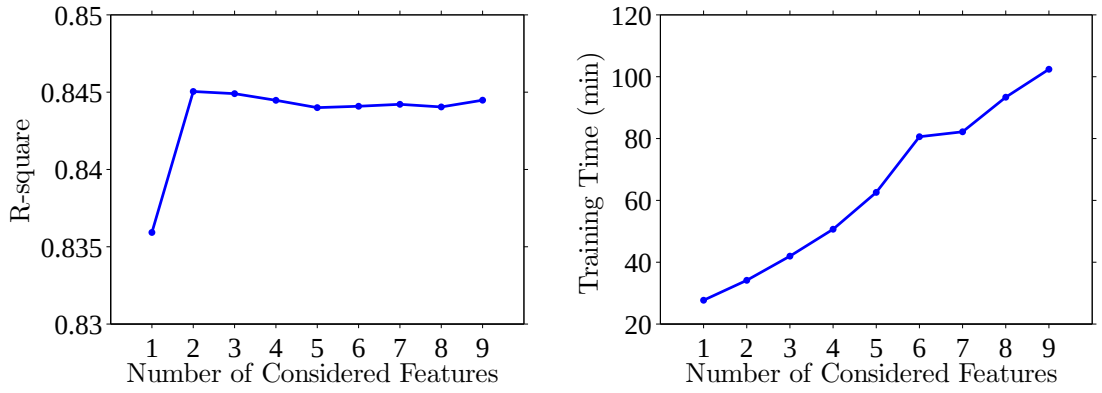
time, we distribute the computational tasks to two machines, namely, an Intel 2.3 GHz workstation and an Intel 1.6 GHz desktop. The difference of the two machines results in incomparable execution time. Therefore, we train the hyperparameters in the following way.



(a) (b)

Figure 4.3: Tuning the number of trees of RF-based algorithm for desktop dataset: (a) R-square and (b) training time.

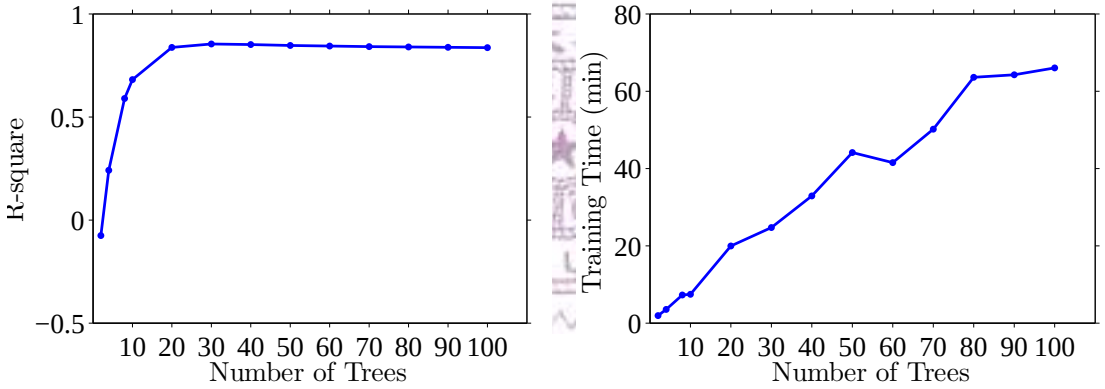
The following are the procedures how we tune the hyperparameters of both algorithms for desktop dataset. For RF-based solution, the hyperparameters include (i) the number of trees t and (ii) the number of considered features f. Namely, $\mathbf{H}_{\text{RF}} = \{t, f\}$. Fig. 4.3 shows the R-square scores and training time with different trees within the range $t = \{2, 4, 8, 10, 20, ..., 150\}$ when $f = 7$. We run the training program on an Intel 1.6 GHz desktop. We find that, when the number of trees increases, the R-square scores do not improve much, and the training time keeps increasing. Since when $t = 80$, the R-square score is slightly higher than the others, we manually set t as 80. We then fix t and vary the number of considered features within the range $f = \{1, 2, ..., 9\}$. We find that the number of considered features do not have obvious effects on R-square results (see fig. 4.4). We then choose to consider all the features, namely, $f = 9$. That is, the optimal



(a) (b)

Figure 4.4: Tuning the number of considered features of RF-based algorithm for desktop dataset: (a) R-square and (b) training time.

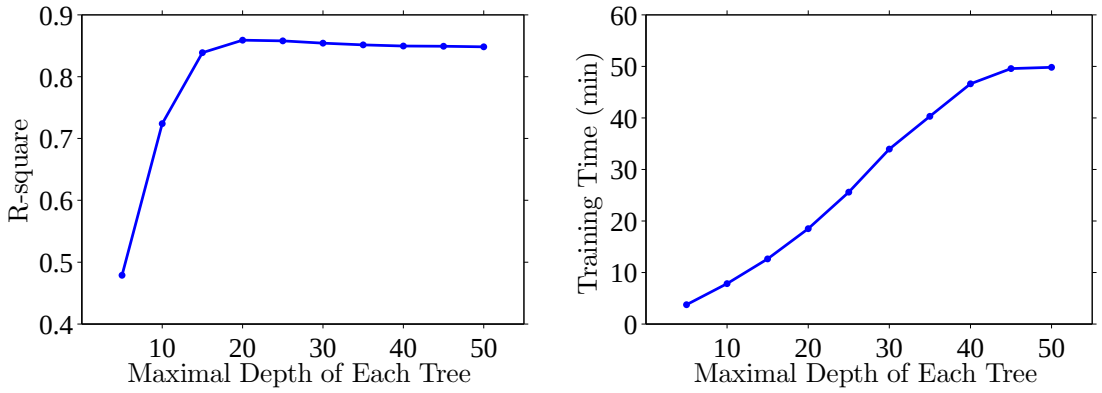
hyperparameters for $\mathbf{D}_{\text{de}}$ is $\mathbf{H}_{\text{RF}} = \{t = 80, f = 9\}$.



(a) (b)

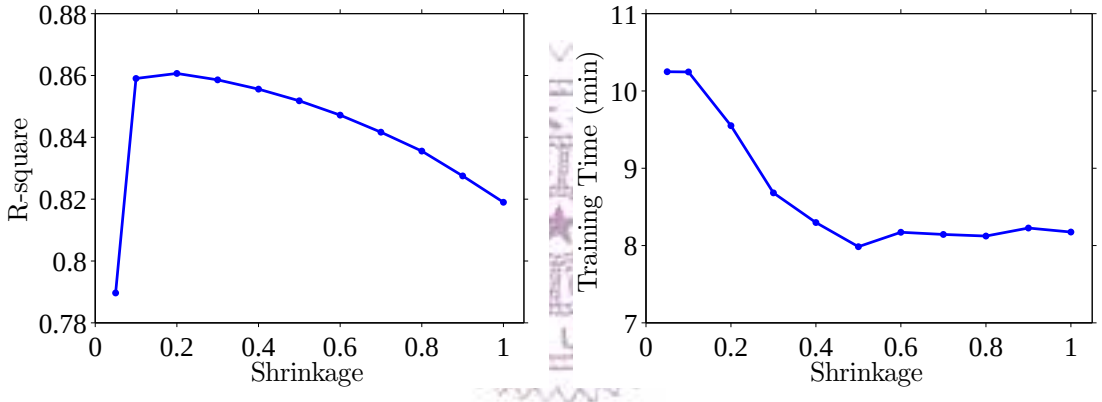
Figure 4.5: Tuning the number of trees of GB-based algorithm for desktop dataset: (a) R-square and (b) training time.

For GB-based solution, the hyperparameters include (i) the number of trees n, (ii) the maximal depth of each tree d, and (iii) the shrinkage s (i.e., the learning rate). Namely, $\mathbf{H}_{\text{GB}} = \{n, d, s\}$. Fig. 4.5 shows the R-square scores and training time with different trees within the range $t = \{2, 4, 8, 10, 20, ..., 100\}$ when $d = 30$ and $s = 0.1$. We run the training program on an Intel 2.3 GHz workstation. The number of trees $n = 30$ performs the best. Therefore we fix the number of trees as 30 and vary the rest two hyperparameters within the ranges $d = \{5, 10, ..., 50\}$ and $s = \{0.05, 0.1, 0.2, ..., 1\}$. Following the same procedure, we find that the maximal depth $d = 20$ and the shrinkage $s = 0.2$ leads to the optimal R-square scores (see fig. 4.6 and 4.7, respectively). That is, the optimal hyperparameters for $\mathbf{D}_{\text{de}}$ is $\mathbf{H}_{\text{GB}} = \{n = 30, d = 20, s = 0.2\}$.



(a) (b)

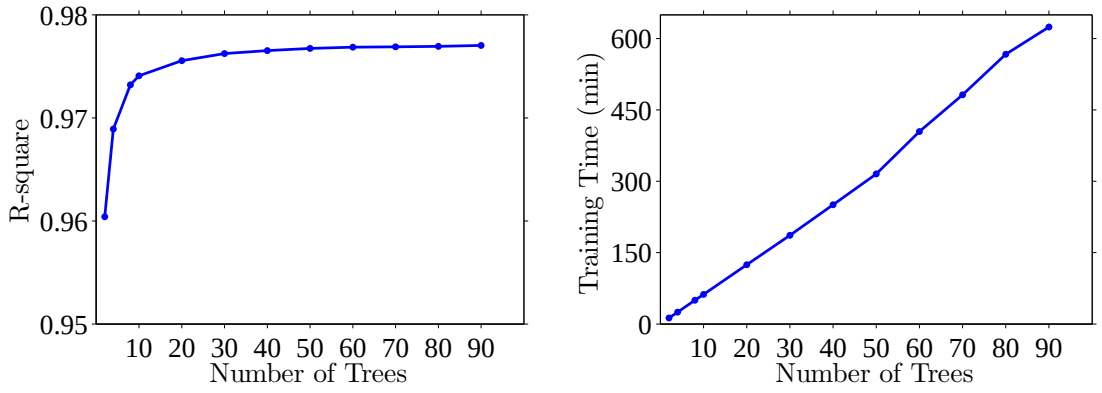
Figure 4.6: Tuning the maximal depth of each tree GB-based algorithm for desktop dataset: (a) R-square and (b) training time.



(a) (b)

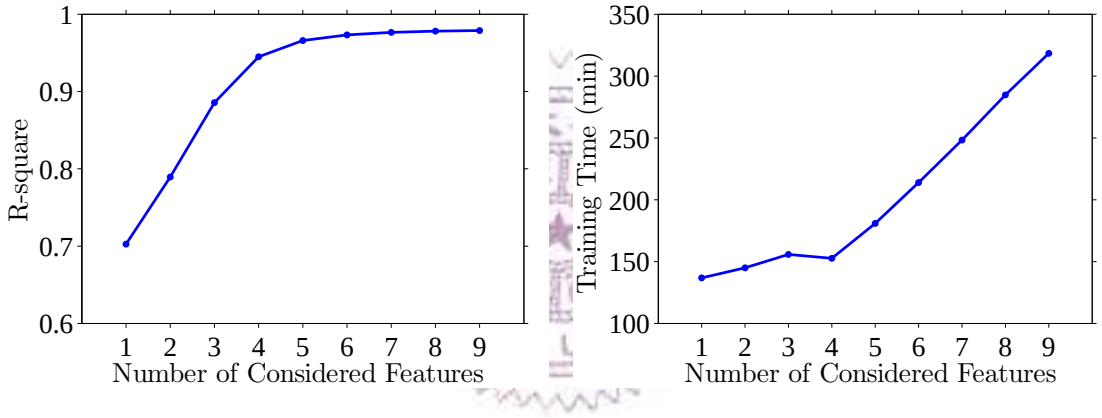
Figure 4.7: Tuning the number of considered features of GB-based algorithm for desktop dataset: (a) R-square and (b) training time.

We tune the hyperparameters of both algorithms for datacenter dataset. Fig. 4.8 shows the R-square scores and training time with different trees within the range $t = \{2, 4, 8, 10, 20, ..., 90\}$ when $f = 7$. We run the training program on an Intel 1.6 GHz desktop. The results are similar with that of desktop dataset. When the number of trees increases, the R-square scores do not improve much, and the training time keeps increasing. Since when $t = 40$, the R-square score is slightly higher than the others, we manually set t as 40. We then fix t and vary the number of considered features within the range $f = \{1, 2, ..., 9\}$. The R-square scores keep improving when the number of considered features increases. Although the improvement of the R-square score slows down after the number of considered features reaches 5 (see fig. 4.9). We then choose to consider all the features, namely, $f = 9$. That is, the optimal hyperparameters for $\mathbf{D}_{\text{da}}$ is



(a) (b)

Figure 4.8: Tuning the number of trees of RF-based algorithm for datacenter dataset: (a) R-square and (b) training time.



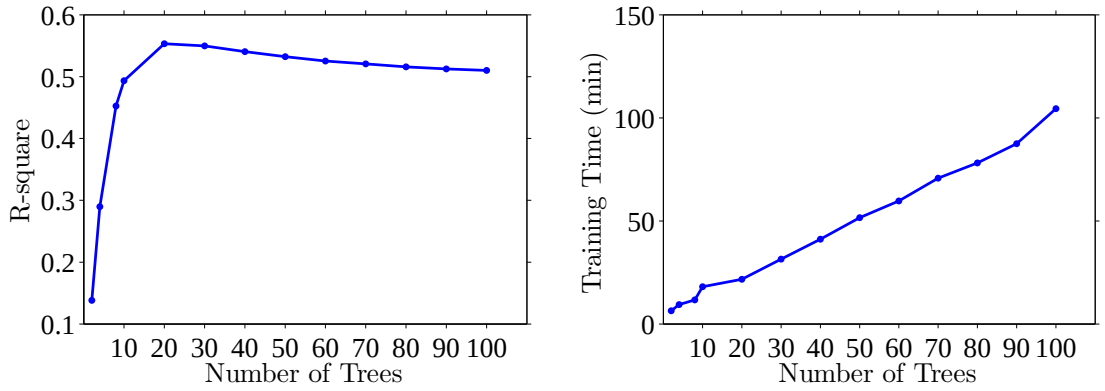
(a) (b)

Figure 4.9: Tuning the number of considered features of RF-based algorithm for datacenter dataset: (a) R-square and (b) training time.

$\mathbf{H}_{\text{RF}} = \{t = 40, f = 9\}$.

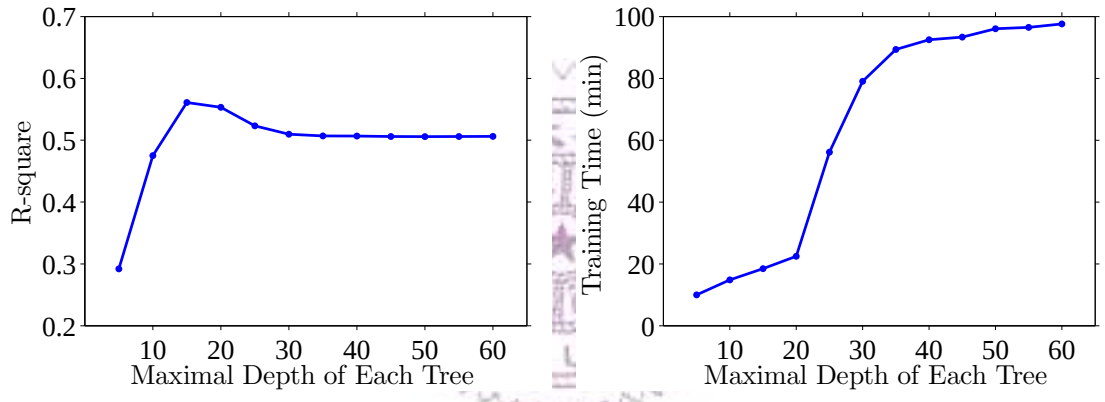
For GB-based solution, we perform the training procedure similar to that of desktop dataset. Fig. 4.10 shows the R-square scores and training time with different trees within the range $n = \{2, 4, 8, 10, 20, ..., 100\}$ when $d = 20$ and $s = 0.1$. We run the training program on an Intel 2.3 GHz workstation. The number of trees $n = 20$ performs the best. Therefore we fix the number of trees as 20 and vary the rest two hyperparameters within the ranges $d = \{5, 10, ..., 60\}$ and $s = \{0.05, 0.1, 0.2, ..., 1\}$. Following the same procedure, we find that the maximal depth $d = 15$ and the shrinkage $s = 0.2$ leads to the optimal R-square scores (see fig. 4.11 and 4.12, respectively). That is, the optimal hyperparameters for $\mathbf{D}_{\text{da}}$ is $\mathbf{H}_{\text{GB}} = \{n = 20, d = 15, s = 0.2\}$.

The hyperparameters of RF-based algorithm for both datasets are showed in 4.2, while



(a) (b)

Figure 4.10: Tuning the number of trees of GB-based algorithm for datacenter dataset: (a) R-square and (b) training time.



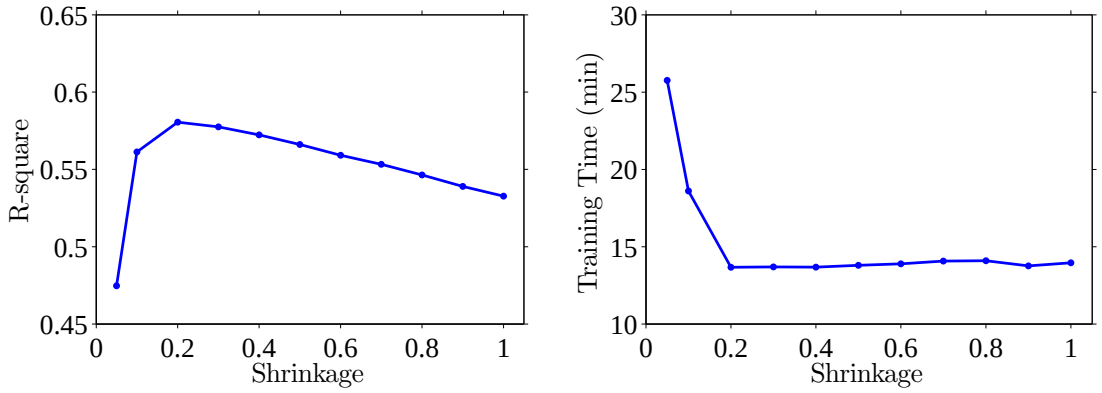
(a) (b)

Figure 4.11: Tuning the maximal depth of each tree GB-based algorithm for datacenter dataset: (a) R-square and (b) training time.

	Desktop Dataset, $\mathbf{D}_{\text{de}}$	Datacenter Dataset, $\mathbf{D}_{\text{da}}$
Number of Trees, t	90	40
Number of Features, f	9	9

Table 4.2: The optimal hyperparameters of RF-based algorithm for desktop dataset and datacenter dataset.

that of GB-based algorithm for both datasets are showed in 4.3. In the rest of the thesis, we applied the hyperparameters derived from the tuning process mentioned above if not otherwise specified. We acknowledge that those derived values are *data-driven*. When the characteristics of the data dramatically change, it is necessary to re-tune the hyper-parameters. Moreover, when the fog provider applies new machine learning approaches, it is necessary to fine-tuned the hyperparameters following above procedure. Note that



(a) (b)

Figure 4.12: Tuning the number of considered features of GB-based algorithm for datacenter dataset: (a) R-square and (b) training time.

	Desktop Dataset, $\mathbf{D}_{\text{de}}$	Datacenter Dataset, $\mathbf{D}_{\text{da}}$
Number of Trees, n	30	20
Depth of Trees, d	20	15
Shrinkage, s	0.2	0.2

Table 4.3: The optimal hyperparameters of GB-based algorithm for desktop dataset and datacenter dataset.

we tune the hyperparameters for the two datasets separately, because they are collected in different settings and from completely unrelated object groups. For each dataset, we combine the traces of all the nodes/users as one dataset. Since we assume that there may be relations of the usage patterns of the individuals. For example, in the desktop dataset, most participants are students. People with geographical proximity, the same career, or other similarities may have similar usage pattern. These information can also be encoded as a feature and be used as the input for the machine learning algorithms. Collecting this kind of information for predictor training is a part of our future work. Note that there are more hyperparameters that can be tuned. We choose the several ones that we consider the most important as example. Note that we also try tuning the hyperparameters for the NN-based algorithm. However, the results do not show obvious trends. Hence we empirically choose to use one single hidden layer, 7 hidden neurons and the learning rate of 0.7, and do not show the procedures in the thesis.

Chapter 5

Data-Driven Simulations

In this section, we show the setup and our simulation results.

5.1 Setup

We implemented a simulator of our multimedia fog computing platform using Java.

Feature	Mean	Std.
CPU Usage (%)	19.7	11.7
RAM Usage (KB)	380.7	147.5
Number of Frames	113.9	76.7
Number of Polygons	63512.6	332868.8
Image Size (Pixels)	131161.6	17453.5
Completion Time (s)	104.1	194.2

Table 5.1: Statistics of the animation rendering dataset.

Our simulator requires the inputs for (a) the available resources of the fog device and (b) the animation rendering jobs.

As we have mentioned in 4.2, we reserve $1/5$ of both dataset as the testing set for the simulation. Datacenter dataset has $2,499,032$ records for testing, while desktop dataset has $593,426$ records. Each record contains the timestamp, the actual amount of available CPU resource, and the predicted amount of available CPU resource. Since the start recording time is different for each node in both dataset, we use Poisson process with a mean arrival rate $\lambda = 30$ minutes to generate the device arrival time. During each simulation, each device will be assigned its arrival time generated by the Poisson process. For one device, we subtract the first timestamp from the timestamp of every resource record. That is, we shift all the timestamp to let the timestamp of the first record be 0. When the simulation starts, we add the assigned arrival time from the Poisson process to every

timestamp for that device.

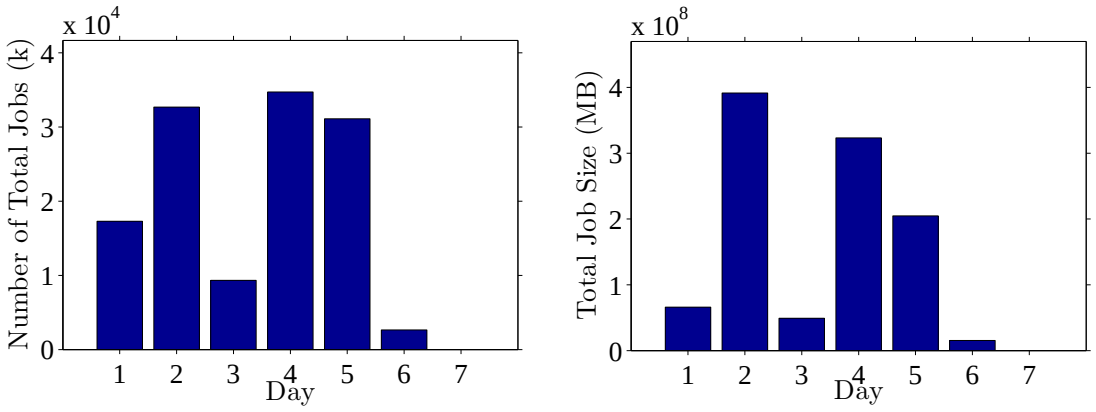


Figure 5.1: Information of arrival jobs on each day: (a) the number of arrival jobs and (b) total size of the arrival jobs.

We apply an *animation rendering dataset* from the collaborated fog rendering company (the company name is withheld due to a non-disclosure agreement) for the simulation. There are $127,791$ records collected between September and November 2015. Each record is a rendering job submitted from an animation studio. Each record contains (a) detailed characteristics of the animation rendering job, such as the number frames and the number of polygons, (b) resource usage information, such as the average CPU usage and consumed GFLOPS, and (c) job processing statistics, such as the timestamp of submitting the job, the timestamp of receiving the completed job, and the estimated monetary cost of the job. Table 5.1 shows some sample statistics of the animation rendering dataset.

When performing the simulation, we found that there are too few jobs arrived on a day. This may be because the company is just newly started. However, we want to simulate a higher loaded scenario. Hence, we adjust the date of the arrival time to concentrate all the jobs to arrive within 7 days. For example, a job arrived at September 1st and another arrived at September 8th are modified to arrive at the same day with their original arrival timing (HH:mm:ss) remained. Moreover, we increase the size of every job by 100 times. Fig. 5.1 are the information of arrival jobs on each day of the adjusted animation rendering dataset. Fig. 5.1(a) shows the number of arrival jobs on each of the 7 days. As mentioned before, there are $127,791$ jobs in total. With adjustment, there are averagely $18,255.9$ jobs arrived at a day, where day 4 has the most jobs with $34,731$ ones, and day 7 has no job. Fig. 5.1(b) shows the total size of the arrival jobs on each day. Although day 4 has the most jobs, day 2 has the largest total size of the jobs, i.e., $382,463.4$ GB.

To sum up, the following information are from the datasets: (a) the timestamp of the available resource record, (b) the arrival time of the rendering jobs, and (c) the job size.

Besides the proposed RF-based, GB-based, and NN-based algorithms, we implement a perfect yet unrealistic Oracle algorithm. The Oracle algorithm schedules the jobs according to the actual available resources. It is used as an upper bound of the performance for the prediction algorithms.

We implement the efficient *Earliest Start Schedule* (ESS) [16, Chapter 3.2] algorithm. Our job scheduler batches the arrived jobs every day, schedules them at 23:59, and starts processing them in the next day.

The following are the performance metrics we considered in our simulations.

- **Deviation:** The deviation between the predicted amount of available resource and the actual amount of available resource.
- **Completed jobs ratio:** The ratio of the number of completed rendering jobs to the number of total rendering jobs.
- **Makespan:** The total time to complete a set of rendering jobs submitted by the fog user. The makespan of a rendering job includes the time for both processing and waiting.
- **Number of failed jobs:** The number of rendering jobs that are failed. Jobs that are not completed before a day ends or before the assigned device leaves the system (i.e., the device turns off or loses the network connection) are considered as failed and will not be rescheduled in our simulation.
- **Normalized CPU consumption:** The average CPU consumption normalized to that of the Oracle.

We run the simulation 10 times for the implemented solutions, namely, RF-based, GB-based, and NN-based, and Oracle algorithms. We calculate the average simulation results and present the 95% confidence intervals whenever applicable.

5.2 Results

Since there is no job on day 7, we show the simulation results between day 1 and 6 for all the performance metrics.

NN-based algorithm performs the most accurate prediction for both datasets. Fig. 5.2 shows the deviation of RF-based, GB-based, and NN-based algorithms for both dataset. For both datasets, NN-based algorithm performs the most accurate prediction, namely, the lowest deviation. It achieves 6.08% and 2.00% deviation in average for the desktop and datacenter datasets, respectively, which means the prediction results are very close to the actual amounts of the available resource. RF-based algorithm results in

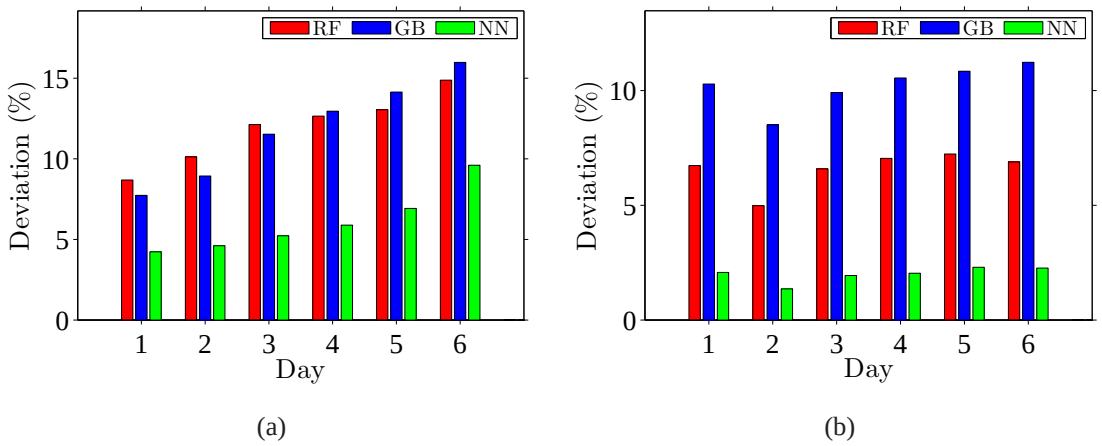


Figure 5.2: Deviation of three solutions for (a) desktop dataset and (b) datacenter dataset.

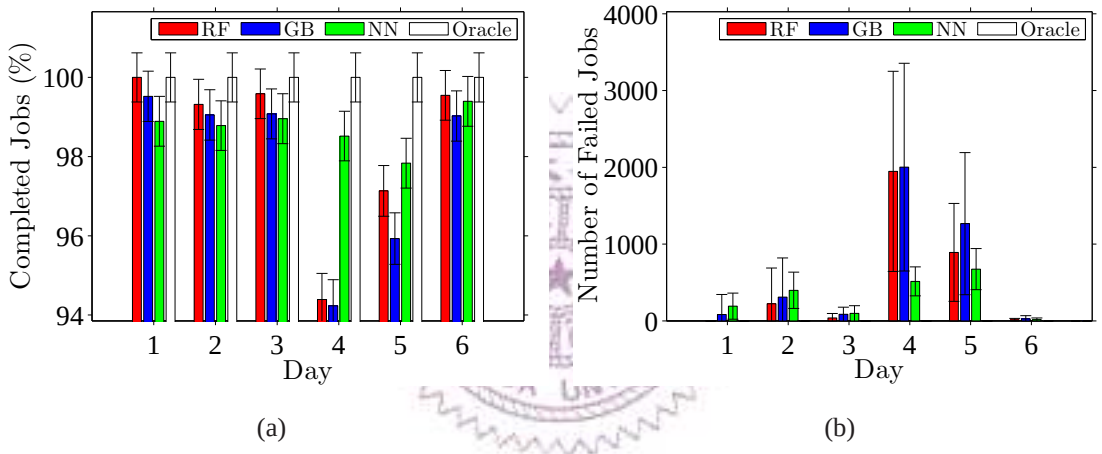
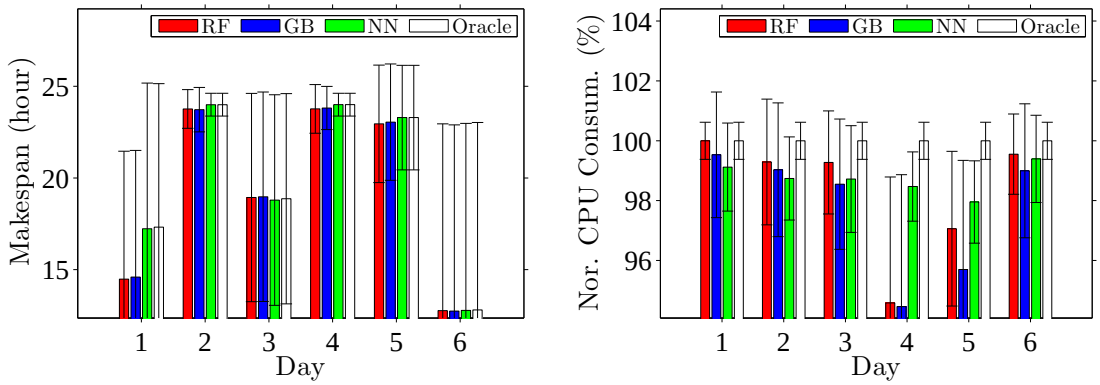


Figure 5.3: Simulation results for desktop dataset: (a) the completed jobs ratio (%) and (b) the number of failed jobs.

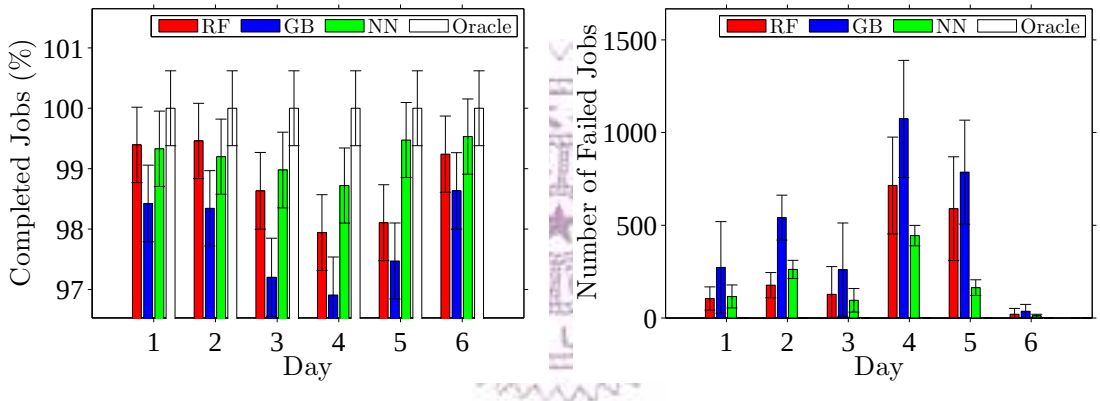
11.91% and 6.58% deviation in average for the desktop and datacenter datasets, respectively. As for GB-based algorithm, it results in 11.88% and 10.22% deviation in average for the desktop and datacenter datasets, respectively. RF-based and GB-based algorithm both perform the worst in day 6 for the desktop datasets, where the deviation goes up to 14.87% and 15.98%, respectively. We note that for the desktop dataset, GB-based algorithm outperforms RF-based algorithm on days 1, 2, and 3, but RF-based algorithm outperforms GB-based algorithm on days 4, 5, and 6. For the datacenter dataset, RF-based algorithm always outperforms GB-based algorithm. This shows that every dataset has its characteristics and that algorithms may perform differently.

More accurate prediction leads to less failed jobs. Since we aggregate all the jobs of the animation rendering dataset within 7 days, there are a large number of jobs arrived every day. While Figs. 5.3(a) and 5.5(a) show that three proposed and the Oracle



(a) (b)

Figure 5.4: Simulation results for desktop dataset: (a) the makespan (hour) and (b) the normalized CPU consumption.

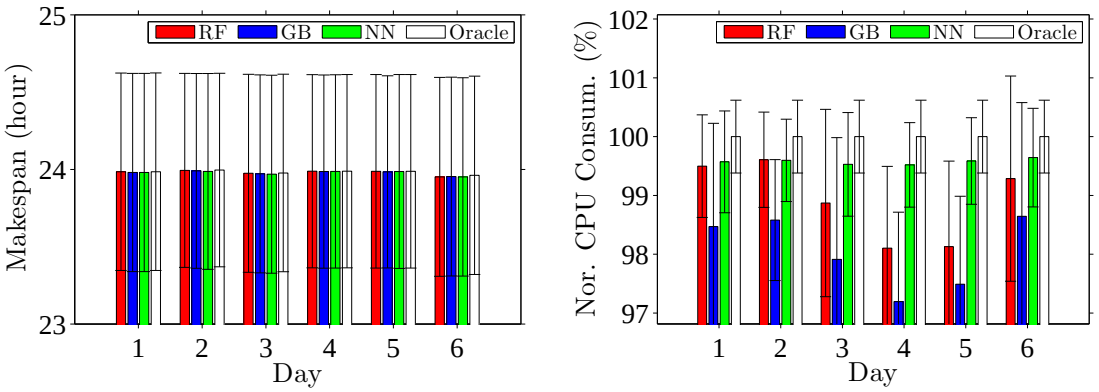


(a) (b)

Figure 5.5: Simulation results for datacenter dataset: (a) the completed jobs ratio (%) and (b) the number of failed jobs.

algorithms all complete 98% or higher of the jobs. Figs. 5.3(b) and 5.5(b) give a closer look into the number of failed jobs. The Oracle algorithm results in very few failed jobs because it checks the actual amount of available resources when scheduling the jobs. It assigns jobs only when the active fog devices have sufficient resources to finish them. From Fig. 5.5(b), the GB-based algorithm which performs the worst prediction leads to the most failed jobs every day. Meanwhile, the NN-based algorithm which performs the best prediction has the least failed jobs for most of the time comparing to the other two algorithms.

From the results of the completed jobs ratio (see Figs. 5.3(a) and 5.5(a)), the makespan (see Figs. 5.4(a) and 5.6(a)), and normalized CPU consumption (see Figs. 5.4(b) and 5.6(b)), we find that our three proposed solutions perform close to the Oracle. This shows that our



(a) (b)

Figure 5.6: Simulation results for datacenter dataset: (a) the makespan (hour) and (b) the normalized CPU consumption.

solutions can perform well with our fine-tuned hyperparameters.

TSING HUA UNIVERSITY

Chapter 6

Conclusion and Future Work

In order to deal with the increasing demand of various resources and to utilize the upgrading computing devices, we proposed a multimedia fog computing platform. In this platform, the fog provider receives the jobs from the fog users and schedules them to the fog workers/devices. There are three main research problems: (i) prediction of the required amount of resources of the jobs, (ii) prediction of the available resources of the fog devices, and (iii) scheduling the jobs and the fog devices. This thesis focuses on the prediction of the available resources. We adopt three machine learning algorithms, namely, the Random Forest, Gradient Boosting Tree, and Neural Network, and implement them using open source libraries. Since each dataset has its own characteristics, it is necessary to fine-tune the hyperparameters for each dataset when applying the algorithms. We apply two datasets, desktop and datacenter datasets. The former is a self-collected trace lasting for 1 month from the laptops or desktops of 25 real users. The latter is an open dataset lasting for 3 months from 500 machines in a datacenter. We use 80% of both datasets and perform 10-fold cross validation to fine-tune the hyperparameters of the proposed algorithms. The results show that the optimal hyperparameters for both datasets are different. We note that when the fog provider applies new datasets, or when the dataset changes, it is necessary to re-tune the hyperparameters. We implement a simulator and use the rest 20% of both available resource datasets and a real animation rendering jobs dataset to drive our simulator. The simulation results show that: (i) the Neural Network-based algorithm achieves 6.08% and 2.00% deviation in average for the desktop and datacenter datasets, respectively, and (ii) more accurate prediction of the amount of available resources leads to fewer failed jobs.

We are considering the following directions to improve our solution: (i) Collect more information as features. For the collected traces, we may record more information such as the age and career on the premise of not invading people's privacy. Since people in approximate ages or with the same occupation may have similar usage patterns, using these

information as the features may be possible to improve the prediction accuracy. (ii) Design a reliability mechanism. By recording the actual available resource and the individual activities in our platform, we can determine the reliability of the fog worker/device. For a fog device which has regular usage pattern, or for a fog worker who joins and leaves our platform regularly, they can be considered with higher reliability. When scheduling the jobs, fog workers/devices are prioritized to get jobs. (iii) Adopt more machine learning algorithms. So far we adopt three machine learning algorithms for the available resource prediction. There is no definite answer that which solution best fits our problem. We are considering to adopt more state-of-the-art machine learning algorithms such as deep learning. Last but not least, we are going to build up a real testbed of our multimedia fog computing platform. Through a real working system, we may evaluate the feasibility of our proposed solutions.

As we have mentioned in Chap. 1, we aimed at building the multimedia fog computing platform. There are three main research problems for our platform: (i) predicting the required amount of the resources of the jobs from the fog users, (ii) predicting the available resources of the fog devices, and (iii) scheduling the jobs and the fog devices. The previous work of our lab [23] has studied the first problem, where we use the RF-based and GB-based algorithms to predict the completion time of the jobs. The difference between this thesis and the previous work is that, predicting available resources in a future time is a time series prediction problem. Not only do we adopt the algorithms that have been used in the previous work, we further introduce the neural network algorithm in this thesis. The results show that NN-based solution performs better accuracy, which encourages us to adopt more machine learning algorithms that may be suitable for time series prediction in the future. We are going to move forward to the rest of the problems to complete the platform. When the fog provider is scheduling the jobs, there are more problems to consider. For example, since a fog worker may leave the system at any time before he finishes and submits the assigned jobs, it is unreliable to assign a job to only one fog worker. One possible solution is to assign duplicate jobs to multiple fog workers at a time. When assigning more duplicate jobs may increase the reliability, it is, however, wasting the resources of the platform. Optimizing the degree of the duplicate job assignment is one of the problems we need to solve. Another problem is to verify the correctness of the submitted results from the fog devices. In the future, we are going to solve the rest of the problems and build up a prototype of the system to verify our proposed solutions in practice.

Here we look into the idea of the fog computing platform from a higher viewpoint. As aforementioned, we want to ease the burden of the cloud platform. However, we still use the cloud computing resources as a backup plan. That is, we plan to integrate the re-

sources from the cloud, the edge cloud, and the fog. The following are several challenges. First, we need to deal with the uncertainty and heterogeneity of the fog workers/devices. The fog workers have different kinds of usage pattern and habit of using his device. To better scheduling the resources and jobs, it is necessary yet challenging to make predictions, which are the problems studied in [23] and this thesis. The second challenge is the dynamicity of fog users' requests. So far we only use animation rendering as a sample application. When we are considering a unified fog computing platform, the jobs from the fog users can be other kinds of multimedia application. To utilize various personal devices, it is challenging to distribute the jobs and make them executable on various operating systems. The last one is to provide QoS guarantees on the resource limited fog devices. When there are a large amount of requests and many resource limited fog devices, we need to tackle the QoS guarantee problem. Last but not least, we envision the benefits that the fog computing platform can bring to our future, which makes it possible to compute everywhere all over the world.

Bibliography

[1] BitBrains. https://www.bitbrains.nl/solvinity-en.

[2] Disney rendered its new animated film on a 55,000-core supercomputer. https://www.engadget.com/2014/10/18/disney-big-hero-6/.

[3] Microsoft developer network (MSDN). https://msdn.microsoft.com/en-us/.

[4] Pixar by the numbers - from toy story to brave. http://collider.com/pixar-numbers-toy-story-brave/.

[5] Pixar by the numbers - from toy story to monsters university. http://collider.com/pixar-numbers-monsters-university/.

[6] Scikit-learn. http://scikit-learn.org.

[7] SETI@home. http://setiathome.berkeley.edu/.

[8] XGBoost. https://github.com/dmlc/xgboost.

[9] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H. Lin. *Learning from data*, volume 4. AMLBook Singapore, 2012.

[10] S. Akioka and Y. Muraoka. Extended forecast of cpu and network load on computational grid. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 765–772. IEEE, 2004.

[11] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[12] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[13] A. Andrzejak, D. Kondo, and D. P. Anderson. Ensuring collective availability in volatile resource pools via forecasting. In *Managing Large-Scale Service Deployment*, pages 149–161. Springer, 2008.

[14] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proc. of ACM Workshop on Mobile Cloud Computing (MCC)*, pages 13–16, Helsinki, Finland, August 2012.

[15] J. Brevik, D. Nurmi, and R. Wolski. Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 190–199. IEEE, 2004.

[16] P. Brucker and S. Knust. *Complex Scheduling*. Springer, 2012.

[17] M. Carvalho, R. Miceli, P. D. M. Jr, F. Brasileiro, and R. Lopes. Predicting the quality of service of a peer-to-peer desktop grid. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 649–654. IEEE Computer Society, 2010.

[18] C. Dabrowski and F. Hunt. Using markov chain analysis to study dynamic behaviour in large-scale grid systems. In *Proceedings of the Seventh Australasian Symposium on Grid Computing and e-Research-Volume 99*, pages 29–40. Australian Computer Society Inc., 2009.

[19] S. Di, D. Kondo, and W. Cirne. Google hostload prediction based on Bayesian model with optimized feature combination. *Journal of Parallel and Distributed Computing*, 74(1):1820–1832, 2014.

[20] N. D. Doulamis, A. D. Doulamis, A. Panagakis, K. Dolkas, T. A. Varvarigou, and E. Varvarigos. A combined fuzzy-neural network model for non-linear prediction of 3-D rendering workload in grid computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1235–1247, 2004.

[21] J. Friedman, R. Tibshirani, and T. Hastie. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[22] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *2007 IEEE 10th International Symposium on Workload Characterization*, pages 171–180. IEEE, 2007.

[23] H. Hong, J. Chuang, and C. Hsu. Animation rendering on multimedia fog computing platforms. In *IEEE 8th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016.

[24] B. Javadi, D. Kondo, J. Vincent, and D. P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@ home. *Parallel and Distributed Systems, IEEE Transactions on*, 22(11):1896–1903, 2011.

[25] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[26] D. Kondo, A. Andrzejak, and D. P. Anderson. On correlated availability in internet-distributed systems. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 276–283. IEEE Computer Society, 2008.

[27] S. J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, Inc., 1995.

[28] K. Ramachandran, H. Lutfiyya, and M. Perry. Decentralized approach to resource availability prediction using group availability in a p2p desktop grid. *Future Generation Computer Systems*, 28(6):854–860, 2012.

[29] B. Rood and M. J. Lewis. Grid resource availability prediction-based scheduling and task replication. *Journal of Grid Computing*, 7(4):479–500, 2009.

[30] L. Sarmenta. *Volunteer computing*. PhD thesis, Massachusetts Institute of Technology, 2001.

[31] L. Shang, Z. Wang, X. Zhou, X. Huang, and Y. Cheng. TM-DG: a trust model based on computer users' daily behavior for desktop grid platform. In *Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing*, pages 59–66. ACM, 2007.

[32] L. Vaquero and L. Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, October 2014.

[33] Y. Wu, Y. Yuan, G. Yang, and W. Zheng. Load prediction using hybrid model for computational grid. In *2007 8th IEEE/ACM International Conference on Grid Computing*, pages 235–242. IEEE, 2007.

[34] D. Yang, J. Cao, C. Yu, and J. Xiao. A multi-step-ahead CPU load prediction approach in distributed system. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 206–213. IEEE, 2012.

[35] Y. Yuan, Y. Wu, G. Yang, and W. Zheng. Adaptive hybrid model for long term load prediction in computational grid. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 340–347. IEEE, 2008.