Decision Support for Computational Offloading by Probing Unknown Services

Christian Meurisch*, Julien Gedeon*, The An Binh Nguyen†, Fabian Kaup‡, Max Mühlhäuser*
Technische Universität Darmstadt, Hochschulstraße 10, D-64289 Darmstadt, Germany
*Telecooperation Lab, {meurisch,gedeon,max}@tk.tu-darmstadt.de
†Multimedia Communications Lab, the.an.binh.nguyen@kom.tu-darmstadt.de
‡Peer-to-Peer Systems Engineering, fabian.kaup@ps.tu-darmstadt.de

***Abstract*—Mobile Cloud Computing leverages resourceful data centers that are distant (aka the cloud) or closely located (aka edge servers) for computational offloading to overcome resource limitations of modern mobile systems like smartphones or IoT devices. Many research works investigate context-aware offloading decision algorithms aiming to find the best offloading system at runtime. However, all approaches require prior knowledge of the offloading systems or a running service profiler on the backend system. In this paper, we present a novel approach that overcomes this issue by first probing available unknown services such as nearby cloudlets or the distant cloud, and networks in an energy-efficient way at runtime to make better offloading decisions. For that, we investigate a probing strategy to assess these unknown services by offloading micro tasks and accurately predicting the performance for larger offloading tasks using regression models. Our evaluation on three algorithms with different time complexities shows that we achieve high prediction accuracies up to 85.5%, already after probing of two micro tasks running in the range of few milliseconds. To the best of our knowledge, this is the first supplement approach for offloading decision support that can handle unknown third-party services requiring no prior knowledge about these offloading systems and making no assumptions for real-world deployments.**

***Index Terms*—mobile cloud computing, computational offloading, decision support, micro-benchmarking, probing**

I. Introduction

Mobile devices like smartphones featured with various sensors and high connectivity become more and more popular over recent years [1]. However, such mobile systems are resource-constrained in view of computational power, storage and battery life due to their small form factor and mobility. Especially, the battery lifetime is still a major issue.

To address these challenges most applications rely on mobile cloud computing (MCC), i.e., resource-intensive tasks are offloaded to powerful servers to save energy [2], [3]. These servers are distant (aka the cloud) or closely located (aka edge servers) to the users. The benefits and drawbacks are obvious. While the cloud is highly available, provides high computational power and has a global view, latency and network traffic are the downsides. In contrast, edge servers like cloudlets provide fast network connection and low latency as well as reduce the network traffic through their proximity to the mobile devices [4]. Depending on the cloudlet deployment (e.g., hosted by ISP, local business or private households), performances and range restrictions are varying [5]. In addition, the context of a mobile device, e.g., locations or network conditions, often changes throughout the day, i.e., the mobile device sees several - partially unknown - offloading systems in daily life.

To find the best offloading system considering the mobile device context, many research works investigate and propose offloading strategies or algorithms to support the decision making process *when and where to offload resource-intensive tasks* [6], [7]. However, all existing approaches require either prior knowledge of the offloading system or a running profiler on the backend system. In real-world deployments, these requirements or assumptions are not given or limited, e.g., when the mobile device discovers unknown computing services hosted by nearby range-restricted cloudlets.

In this paper, we overcome this issue of assessing unknown computing services by probing them at runtime. In this context, *probing* is the process of offloading micro tasks to assess the network capabilities and the backend system. Based on this micro benchmarks, our approach then estimates the performance and the costs of large offloading tasks using regression models to provide a decision support for computational offloading. For that, we investigate the probing strategy in detail: *how many and which micro tasks do we need to offload to accurately estimate the performance of larger offloading tasks?* In the next step, we evaluate the energy consumption or overhead costs required to achieve accurate estimations. Our results show that it is possible to predict the runtime performance of unknown services with an accuracy up to 85.5%, already after two micro tasks running in a range of few milliseconds. We also show that additional performance samples further improve our regression models and reduce the prediction error. Existing approaches can benefit from our probing approach and results for their real-world deployment since they no longer need prior knowledge about the offloading systems. Probing unknown services at runtime provides required parameters (e.g., expected processing time or network delay) for their decision algorithms.

In summary, the contributions of this paper are twofold:

- We present our novel approach to first probe unknown services by micro-benchmarking at runtime for accurately assessing the performance for larger offloading tasks using regression models.
- To show the feasibility and applicability of our approach, we evaluate a probing strategy on three algorithms with

different time complexities and also discuss the overhead costs of probing in terms of prediction accuracy.

The remainder of this paper is organized as follows. We first provide an overview of the related work and existing solutions (Section II). After listing the existing works, we highlight current issues and describe the need for offloading strategies and probing of unknown services (Section III). In Section IV, we present and describe our approach for probing unknown services to support offloading decisions. Section V specifies the experimental setup. The paper closes with the result reporting and discussion (Section VI) as well as the conclusion (Section VII).

II. Related Work

The need for offloading computational tasks and storage from resource-constrained mobile systems (e.g., smartphones or Internet-of-things devices) introduced *mobile cloud computing* [2], [8], [9], [10] or *cyber foraging* [11], [12] about fifteen years ago. Since then, various offloading approaches regarding networked computing infrastructures (e.g., *cloud computing* [13], *cloudlets* [4], [14], [15], *fog computing* [16], [17]), offloading techniques (e.g., MAUI [7], CloneCloud [18]), and offloading strategies (e.g., [6], [19]) were proposed to overcome individual issues and find a tradeoff between performance, latency and network traffic.

A. Offloading Infrastructures

In [5], we already compare different existing offloading infrastructures in terms of computational and network performance. In this section, we revisit these infrastructures and describe individual benefits and drawbacks [3].

1) Mobile Computing: Mobile devices are able to process data locally without latency issues or offload to other nearby mobile devices over a P2P network [20]. However, due to their small form factor and high mobility mobile devices have limited resources, e.g., battery life, storage and computational power [21]. Especially, local processing of resource-intensive tasks drains the battery very fast [5].

2) Cloud Computing: Resource-intensive tasks are offloaded via the Internet from mobile devices to centralized resourceful data centers, the *cloud*. The cloud is a highly scalable computing and storage infrastructure hosted by cloud providers (e.g., Google, Amazon, DigitalOcean or Salesforce) [13]. A cloud serves and stores personal data of hundreds or thousand users at a time. Security, privacy and trust are highly critical points. However, clouds are distant to mobile users and have too long WAN latency for responsive applications. But they are well-suited for applications requiring a global view or historical data. Moreover, only few data centers are deployed in the world with high building and operational costs.

3) Cloudlet: Resource-intensive tasks can also be offloaded from mobile devices via wireless technologies (e.g., WLAN) to a *cloudlet*, a proximate decentralized computing infrastructure hosted by a local business (e.g., coffee shop) [4] or Internet Service Providers (ISP) [22]. It provides low latency due to its proximity to mobile users and high bandwidth. Thus, cloudlets are well-suited for real-time responsive applications like face, gesture or object recognition that only need temporary caches [23]. Cloudlets only need to serve few users at a time. However, a large-scale deployment of current approaches is difficult due to their range restrictions and their relative high costs. In [5], the authors propose upgrading wireless home routers to enable an economic large-scale deployment of cloudlets in urban environments. This concept sets up access points (AP) as cloudlet. On the contrary, Xu et al. study the strategic placements of cloudlet viewed as 'data-center in a box' at some AP locations in large-scale Wireless Metropolitan Area Network (WMAN) [24].

B. Offloading Techniques

In this section, we list the different offloading techniques ranging from data offloading and consuming computing services over code offloading to entire virtual machine migration maintaining program states.

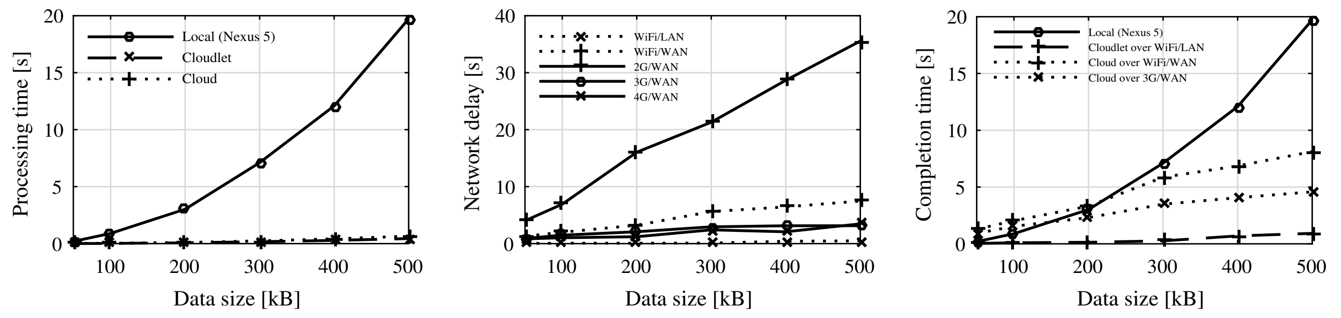
1) Data Offloading: The most common used offloading technique is data offloading (e.g., to the cloud or cloudlets) with consuming of remote computing services [5]. Especially in mobile sensing [1] or anticipatory mobile computing [25], only the data are offloaded, stored and analyzed remotely to build recognition or prediction models.

2) Method-level Code Offloading: Resource-intensive applications or algorithms (e.g., face or voice recognition) can be (partially) offloaded. One popular framework is *MAUI* which provides method-level code offloading [7]. However, MAUI does not address the scaling of execution in cloud. The *ThinkAir* framework, which also performs method-level computation offloading, overcomes this issue by parallelizing method execution using multiple virtual machine (VM) images on the backend side [26]. In [27], Yang et al. introduce a technique of compiler code analysis to identify important code regions for offloading to reduce the network traffic.

3) Virtual Machine Migration: Instead of only offloading data or code, the authors of *CloneCloud* presents a flexible architecture that enables unmodified mobile applications to run in an application-level cloned VM on a computational cloud [18]. This technique ensures that the application runs with same operational environment than on the mobile system. However, running a cloned VM on the offloading system requires high setup costs which is not suitable for stateless offloading systems like cloudlets, where mobile devices are only shortly connected.

C. Offloading Strategies

Given the variety of offloading possibilities, the mobile device needs to decide *where and when to offload resource-intensive tasks* or *is it more energy-efficient to process the task locally*. Early approaches simply execute the task locally with a timeout [28]. If the computation is not completed within this timeout, the task is offloaded to a server. Over the years, the decision algorithms get more complex since they consider contextual parameters. For example, Zhou et



(a) Performance issues (b) Network issues (c) Tradeoff finding issue

Fig. 1: Issues in Mobile Cloud Computing: *performance*, *communication*, and *tradeoff finding*

al. present a context-aware offloading decision algorithms aiming to provide code offloading at runtime [6]. For that, the algorithm selects the wireless medium and the offloading resources which are most qualified for the tasks considering the actual mobile device context. Other offloading algorithms maximize the system throughput based on an admission cost model [19], improve the offloading performance by a risk-controlled decision [29], or by task delegation [30]. In [31], Geng et al. develop an energy-efficient offloading strategy using Dijkstra's algorithm to find the optimal decision in cellular networks.

However, all existing approaches require prior knowledge of the offloading system or a running profiler on the backend system. In real-world deployments, these requirements or assumptions are not given or limited, e.g., when the mobile device discovers unknown computing services hosted by nearby range-restricted cloudlets.

III. The Need For Offloading Strategies and Probing of Unknown Services

We identify three issue groups which need to be considered in terms of mobile cloud computing: (1) *limited mobile resources*, e.g., battery life, storage, computational power, (2) *communication issues*, e.g., latency, bandwidth, network traffic, and (3) *remote processing issues*, e.g., security, privacy, ownership, scalability, deployment and operational costs [5].

Mobile cloud computing aims to overcome the first issue of limited mobile resources by offloading resource-intensive tasks but then it faces with communication and performance issues. Figure 1 illustrates these both issues on sample offloading tasks where larger data sizes indicate more resource intensive processing [5]. In comparison to the distant cloud or nearby cloudlets, the performance of local processing is much lower, especially for high resource-intensive tasks (cf. Fig. 1a). If only a few users are connected to the offloading systems, we can assume that performance remains roughly constant for the same offloading task. However, we cannot assume that this is true for the network conditions since the user carrying his smartphone is highly mobile and has different connectivities based on his location (cf. Fig. 1b). While the processing times on the cloud are normally lower than the local processing times, there exists cases where the total completion times are much lower for local processing, e.g., if the mobile device has to offload the task via 2G connection (cf. Fig. 1c). Thus, context-aware offloading algorithms that make a offloading decision at runtime are required to find a tradeoff between processing time and network delay (e.g., [6]). Some other factors like energy consumption, resource usage, or user's mobility are also an issue in today's offloading strategies (for more details, the reader is referred to [5]).

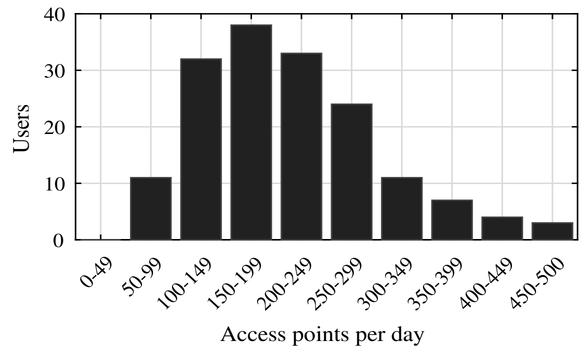


Fig. 2: Issue of *unknown offloading systems* in Mobile Cloud Computing: daily seen access points by users ($>$ 10min)

However, we see an open and important challenge in capturing or estimating performance characteristics of unknown computing services. As a consequence, these third-party services can be considered in existing offloading schemes (e.g., [6]) from now on, and give a complete offloading decision support in real-world deployments.

To highlight the need of assessing unknown services, we investigate daily user behaviors. For that, we collect a large dataset of $22,361$ unique access points spatially covering the whole city of Darmstadt, where our university is located. In addition, we conduct a four-week user study with 163 students living in Darmstadt to get mobility patterns with over 14 million unique location values of them [32]. Correlating these two comprehensive data sets, Figure 2 shows *how many different access points a participant saw longer than 10 minutes in his daily life*. An average user sees about 214 ± 89 access points in his daily life. Applying concepts like [5], [24], we assume a small proportion of these access points will also provide ad-hoc cloudlet functionalities in future. This small

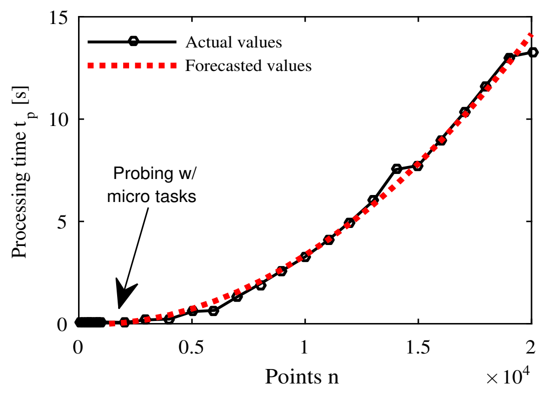


Fig. 3: Our approach: *probing* of computing services with micro tasks to estimate the performance for larger tasks

thought experiment supported by a real-world study should illustrate how many possible offloading systems with different characteristics a user is able to access throughout the day. It is obvious that most of these provided computing services are *unknown* to the user's mobile device, i.e., the mobile device does not know any performance characteristics about the system and the network. Thus, these available offloading systems would not be considered by conventional offloading strategies such as [6].

IV. Our Approach: Probing

In this paper, we overcome this issue of assessing unknown computing services by probing them at runtime. In this context, *probing* is the process of offloading micro tasks to assess the network capabilities and the backend system. Based on this micro benchmarks, our approach then estimates the completion time t_c and the cost c of large offloading tasks to provide a decision support for computational data offloading.

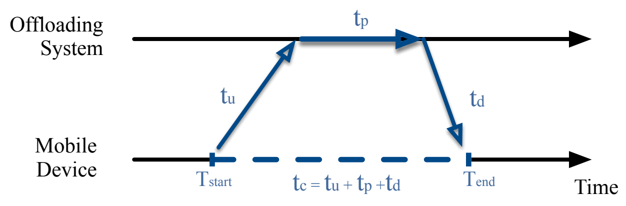


Fig. 4: Timing diagram between mobile device and backend system for task offloading

Figure 3 drafts our approach of probing the performance of backend computing systems (e.g., cloudlet or cloud) without prior knowledge or additional profilers running on them. The algorithm works as follows: *first*, we offload m micro tasks[1] to the unknown system to get some performance samples. Each performance sample, which represents one task offloading, contains the measured completion time t_c (i.e., time range between transmitting the task from the mobile device and receiving the result back) and the measured average energy consumption $\overline{e}$ for this on the mobile device. The completion time $t_c = t_t + t_p$ consists of the network delay t_t for transmitting the task and the processing time t_p for executing the task on the backend system. The network delay $t_t = t_u + t_d$ can be further divided into the upload time t_u of the data and the download time t_d to get the result back (cf. Figure 4). Both the upload and the download times depend on the transmitted data sizes d_u and d_d required for offloading the task, as well as the bandwidths b_u^y and b_d^y of the given network technology y as follows: $t_u = d_u * b_u^y$ or $t_d = d_d * b_d^y$. Since the mobile device can measure the completion time t_c, the data sizes d_u and d_d, as well as the bandwidths b_u^y and b_u^y for the used network technologies, we can calculate an estimated processing time of executing the micro task on the backend system for its current utilization at runtime:

$$t_p = t_c - d_u * b_u^y - d_d * b_d^y \tag{1}$$

Utilizing energy models [33], we can calculate the average energy consumption $\overline{e}$ and determine the energy cost for offloading a single task:

$$c = \overline{e} * t_c \tag{2}$$

The total overhead cost of m micro task samples required for our probing approach can then be calculated as:

$$c_{overhead} = \sum_{i=1}^{m} c_i = \overline{e} * \sum_{i=1}^{m} t_{c_i} \tag{3}$$

Second, we apply a general polynomial regression model with degree k on these m samples:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_k x^k + \epsilon \tag{4}$$

Depending on the algorithm complexity, the degree of the polynomial regression model needs to be adjusted. For instance, given a computing service with complexity of $\mathcal{O}(n)$ the resulting model is a linear regression: $y = a_0 + a_1 x + \epsilon$. Another example is DBSCAN - a density-based clustering algorithm - with an average complexity of $\mathcal{O}(n \log n)$ and a worst case run time complexity of $\mathcal{O}(n^2)$ [34]. We would model this case with a quadratic polynomial or polynomial of the second degree: $y = a_0 + a_1 x + a_2 x^2 + \epsilon$. All in all, our generic approach of applying a general polynomial regression model covers many different computational algorithms with various complexities. The results of applying the regression analysis on the m measured samples are the missing parameters $\{a_i | 0 \leq i \leq k\}$ for the forecasting equation.

In the *third* and last step, we can then provide decision support for existing offloading strategies (e.g., [6]) by using the resulting forecasting equation to estimate the cost and the performance of the unknown service for larger offloading tasks at its current runtime utilization. It is important to note that this approach can be fully automatic executed and

[1]A micro task is a very small offloading task with a completion time in the range of few milliseconds, while the actual offloading tasks may take several seconds due to their larger data sizes.

Model	LG Nexus 5
Processor	Quad-core 2.26GHz Qualcomm Snapdragon 800 (ARM)
Memory	2 GB RAM
Storage	16 GB
OS	Android v5.1.1 (Lollipop)
Power	3.8V, 2300mAh LiPo battery (8.74 Wh)
WLAN	IEEE 802.11 a/b/g/n/ac, dual-band (2.4/5GHz)
Network	GSM (2G) / UMTS (3G) / HSDPA (3.5G) / LTE (4G)

TABLE I: Smartphone specifications

	Cloud	Cloudlet
Processor	2x vCPU@2.4GHz (x64)	4x CPU@2.6GHz (x64)
Memory	2 GB RAM	6 GB RAM
Storage	40 GB SSD	1 TB HDD
OS	Ubuntu 14.04.4 x64	Ubuntu 14.04.4 x64
Distance [km]	~6,200 (NYC,USA)	0.005
Latency [ms] *(WiFi/2G/3G/4G)*	112 / 351 / 300 / 189	14 / - / - / -

TABLE II: Offloading system specifications

be integrated in existing offloading solutions as supplement for handling - currently disregarded - unknown third-party offloading services.

To evaluate our approach and find the best energy-efficient probing strategy, we need to study these challenges:

- How many micro task samples need to be offloaded to accurately estimate larger offloading tasks?
- What is the overhead of *probing* in terms of energy consumption?
- What is the best tradeoff between accuracy and costs for probing unknown computing services?

In the following of this paper, we conduct several experiments to answer these research questions.

V. Experimental Setup

In this section, the experimental setup is described. Studying the probing strategy, our experimental setup simply consists of a mobile device and two offloading systems: a cloud and a cloudlet. Showing the overhead or costs of our probing approach, we also need to consider the several network connections ranging from WiFi to various cellular networks (i.e., 2G, 3G, LTE). We implemented a profiling application to measure the mobile resources (i.e., battery consumption, cpu and memory usage) as well as assess the offloading performance (i.e., processing time and network delay).

A. Hardware

1) Mobile Device: We use a *LG Nexus5* smartphone with quad-core ARM processor (Qualcomm Snapdragon 800) which each core running at 2.26GHz, 2GB memory and 16GB storage (cf. Table I). The operating system is a standard Android 5.1.1 ROM, namely Lollipop. All background services not required for running the operating system are disabled. Nexus5 is equipped with 2300mAh Lithium polymer (LiPo) battery by default. We chose this smartphone since it includes all electronics required for measuring the battery voltage and the current flowing from battery to the device. Thanks to integrated MAX170485 fuel-gauge chip[2] that provides high-accuracy voltage measurements and battery level estimation. It has a resolution of 1.25 mV with an error of 7.5 mV. Accurate enough for our measurement purpose to detect differences between the single offloading use cases. Nexus5 is also equipped with an IEEE 802.11 a/b/g/n/ac wireless transmitter and supports all digital cellular networks ranging from 2G (GSM) to 4G (LTE). Thus, this smartphone is able to offload tasks to the cloud over cellular networks as well as to the cloudlet over wireless technologies [35].

2) Cloud: As cloud backend, we use a DigitalOcean[3] instance hosted in New York City, USA (cf. Table II). The instance provides two compute units at 2.4GHz, 2GB RAM and 40GB SSD storage. The cloud is about $6,200km$ (beeline) faraway from our university (TU Darmstadt, Germany), where we conduct the measurements. This results in a measured latency of $112ms$ or $351/300/189ms$ when the mobile device is connected via WiFi or via cellular networks (2G/3G/4G) to the next Internet access point. Consequently, we can say latency cannot be ignore when talking about distant clouds since these high latencies have a significant impact on the offloading behavior [36]. For more details and benchmark results, we refer to [5].

3) Cloudlet: As cloudlet we use a desktop computer with quad-core x64 processor (Intel Core i5) running each core at 2.6GHz, 6GB RAM, 1TB HDD storage and linux-based operating system (cf. Table II). The same processing code as used for the cloud is also used for the cloudlet. The cloudlet is placed in the near of the mobile device ($\sim 5m$) with one-hop latency ($14ms$) over wireless LAN.

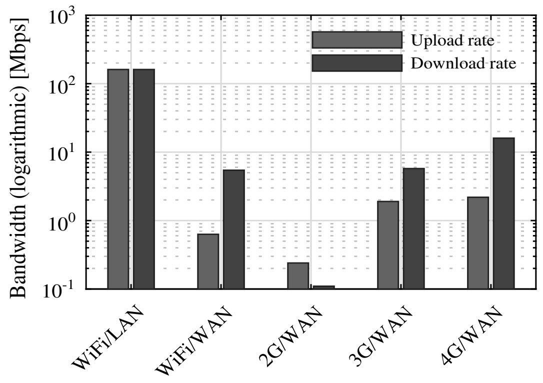


Fig. 5: Network specifications: measured bandwidths of considered networks within our experiments

B. Network Specifications

We consider four network connectivities in our experiments, namely WiFi, 2G, 3G and LTE (4G). Figure 5 shows the measured up- and download bandwidths in our test scenario. As excepted, the bandwidths over WiFi/LAN with low latency

[2]http://www.maximintegrated.com/en/products/power/battery-management/MAX17048.html (accessed 2017-01-10)

[3]https://www.digitalocean.com (accessed: 2017-01-10)

Algorithm	Description	Sample use cases	Complexity		
			best	average	worst
Radix Sort	Sorting algorithm	Time series analysis	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
DBSCAN [34]	Density-based clustering algorithm	Sensor data (e.g., locations) processing	$\mathcal{O}(n\ logn)$	$\mathcal{O}(n\ logn)$	$\mathcal{O}(n^2)$
DFT	Frequency spectrum analysis	Image, audio, or video processing	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

TABLE III: Offloading tasks with different complexities using in our experiments

($14ms$) to our cloudlet are much higher ($160.95 \pm 23.12 Mbps$) than over cellular networks or WiFi/WAN with $112ms$ latency to the cloud. However, we do not achieve the theoretical bandwidth values for any connectivity. Especially, the measured bandwidth values of LTE (up: $2.19 \pm 0.28 Mbps$, down: $16.00 \pm 1.43 Mbps$) are far below than the theoretical ones ($50 - 150 Mbps$).

C. Profiling

Inspired by existing works (e.g., [37]), we implement a lightweight runtime profiler (i.e., an Android app running in the background) which measures three metrics for our microbenchmarks (aka *probing*): *task completion time* consisting of *processing time* and *network delay*. In addition to them, the profiler permanently monitors and logs resource usages: *CPU usage*, *memory usage*, and *energy consumption* on the mobile device. We chose a sampling rate of 300ms for CPU and memory monitoring, and a sampling rate of 50ms - a good, empirical determined balance between accuracy and CPU load - for energy measurements on a Nexus5 [5].

D. Offloading Tasks

To show the feasibility and applicability of our probing approach, we use three sample algorithms with different complexities, namely *Radix Sort*, *DBSCAN*, and *Discrete Fourier Transform (DFT)*, as offloading tasks in our experiments (cf. Table III). These more or less resource-intensive algorithms are often use to process sensor data (e.g., location values, images, audio or video streams) collected by the mobile device. Especially, responsive and latency-critical use cases such as image processing, face or voice recognition, which rely on *Discrete Fourier Transforms* with a high time complexity of $\mathcal{O}(n^2)$, must be nearby offloaded and are part of our evaluation. We also investigate algorithms such as *DBSCAN* with time complexities varying with different inputs of the same size. More precisely, the time complexity of *DBSCAN* is unstable and ranges between $\mathcal{O}(n\ logn)$ and $\mathcal{O}(n^2)$, which represents one of the challenges for estimating the task completion time. As simple algorithm and to show that our approach also works there, we use *Radix Sort* with a stable linear time complexity of $\mathcal{O}(n)$.

E. Accuracy Evaluation Metrics

To evaluate the accuracy of our probing approach, we compare our forecasted values, which are the results of our regression analysis, against actual reference values measured for larger offloading tasks. For each accuracy evaluation, we choose three reference values for offloading tasks whose input sizes are at least an order of magnitude greater than the largest input size of micro tasks. Depending on the complexity of the underlying algorithm, this could make a difference in the processing times of more than one order of magnitude. For instance, if our probing approach offloads two micro tasks with input sizes of 5 and 10, we would use input sizes of 100, 150, and 200 as reference values.

To determine the prediction accuracy of our forecasting method, we use a statistical measure, namely *symmetric mean absolute percentage error*:

$$sMAPE = \frac{\sum_{i=1}^{n} |f_i - a_i|}{\sum_{i=1}^{n} (f_i + a_i)}, \tag{5}$$

where n is the number of reference values, a_i is the actual reference value, and f_i is the forecasted value by our probing approach. We chose this measure to (1) get a relative comparable error, and (2) avoid non-symmetric issues by treating over- and under-forecasts equally [38]. Finally, the prediction accuracy can then be defined as

$$acc = (1 - sMAPE) * 100. \tag{6}$$

We will report following results by calculating an average accuracy, error, and energy consumption over three experimental runs per scenario to reduce measurement errors.

VI. Results

The main contribution of this paper is the study of how to probe unknown systems for providing an accurate offloading decision support. In the following, we present our results in terms of accuracy, overhead and costs for estimating larger offloading tasks.

A. How Many Micro Tasks Need to Be Offloaded to Accurately Estimate Larger Offloading Tasks?

We first investigate how many micro task samples are required to build an accurate regression model, which is able to predict the performance of large offloading tasks. It is important to note that micro tasks containing comparatively few data points and, thus, have completion times in the range of few milliseconds. While actual offloading tasks contain much more data points and have completion times by one to several orders of magnitude.

To show the feasibility of our approach, we rely on a simple reproducible probing strategy applicable to arbitrary underlying processing algorithms: we use a micro task s_0 with data size[4] d_{u_0} that would take about $100ms$ on the mobile device as starting point. For each further micro task s_i, we then

[4]The data size d_u is proportional to the containing sensor data values n, which are the input for the processing algorithms.

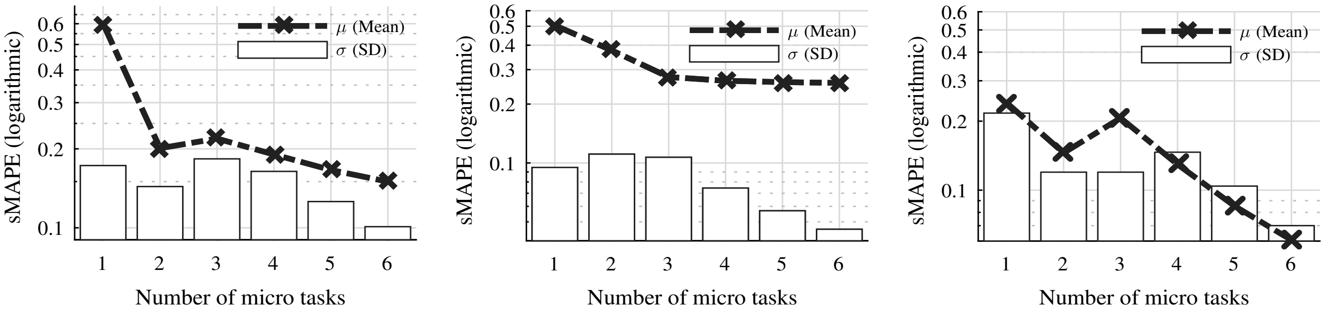


Fig. 6: Relative prediction errors using sMAPE as function of required micro tasks considering three algorithms with different time complexities and five test scenarios, i.e., offloading to the cloudlet and to the cloud over different network technologies

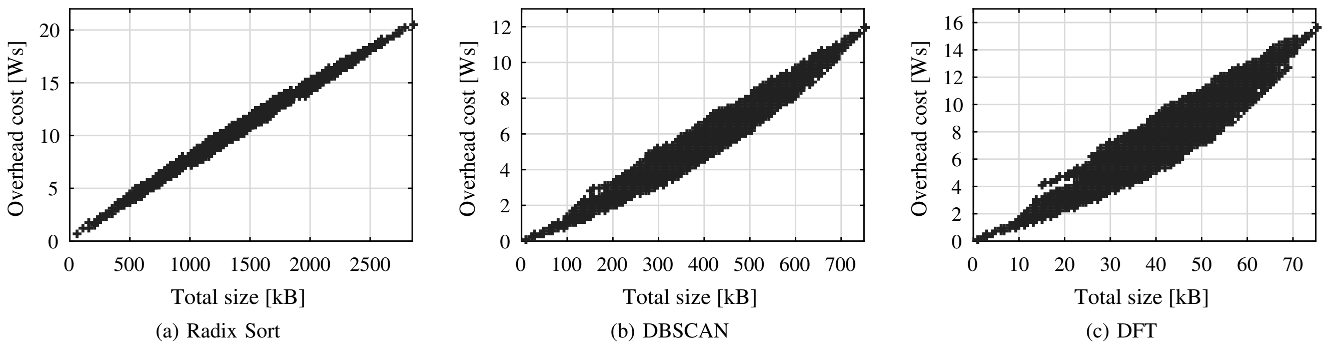


Fig. 7: Total overhead costs of probing over the sum of required micro task data sizes (= total size)

add 50% of the initial data size, i.e., $d_{u_i} = d_{u_0} * (1 + 0.5 * i)$, to obtain samples with an adequate distance for the regression analysis. Since we use the statistical measure *sMAPE* to analyze relative forecasting errors (cf. Equation 5), we can consider the five available test scenarios covering different backend systems and network technologies together, i.e., offloading to the cloudlet via WiFi/LAN as well as offloading to the cloud via WiFi/WAN, 2G/WAN, 3G/WAN, and 4G/WAN (cf. Figure 5).

Figure 6 shows the prediction accuracy results as a function of required micro tasks. In Figure 6a and 6b, we can see that the mean error of the regression model for one micro task sample is very high for *radix sort* (M = 0.601, SD = 0.173) and *DBSCAN* (M = 0.505, SD = 0.095). For *DFT* - an algorithm with quadratic time complexity - the mean error is 0.240 for one micro task sample, which is relative low (cf. Figure 6c). However, the standard deviation of 0.217 is very high.

Notably, the prediction error for *radix sort* decreases significantly by considering two micro task samples (M = 0.201, SD = 0.144). The same is true for both other algorithms, *DBSCAN* (M = 0.376, SD = 0.111) and *DFT* (M = 0.145, SD = 0.120), although the decrease is not as high as for *radix sort*. Considering three or more micro tasks in the regression models, the prediction errors are roughly the same for *radix sort* and *DBSCAN*. Only the *DFT* regression model gets much more accurate when considering up to six micro tasks (M = 0.061, SD = 0.070). We can also see that the variance or standard deviation further decreases, especially by considering six micro task samples, e.g., *radix sort* (M = 0.150, SD = 0.101), and *DBSCAN* (M = 0.257, SD = 0.046).

In conclusion, our probing approach achieves adequately low prediction errors for all three test algorithms by only offloading more than one micro task. More precisely, two or three micro tasks are entirely enough to get accurate prediction results. We can further say that the regression model gets more accurate, more micro tasks are considered, i.e., the mean error and, especially, the variance decrease. However, the more micro tasks are needed to be offloaded, the higher the overhead costs for our probing algorithm.

B. *What are the Overhead Costs of Probing?*

We now investigate the correlation between the number of micro tasks and the overhead costs, which contain the energy consumption of the mobile device during the entire task offloading process (cf. Equation 3).

Figure 7 shows the total overhead costs over the total data sizes, which is the sum of required micro task data sizes. Obviously, the higher the offloaded data size, which needs to be transmitted and processed, the higher the overhead costs. In the case of *radix sort*, this dependency is a clearly linear correlation (cf. Figure 7a).

We see a similar correlation for *DBSCAN* (cf. Figure 7b) and *DFT* (cf. Figure 7c). However, there is a larger spread of the measured data points, which is accounted for by the

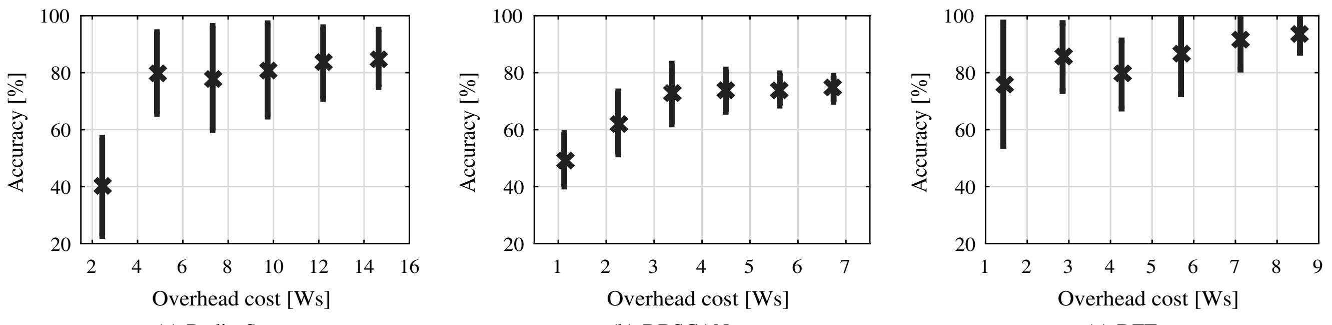


Fig. 8: Tradeoff: prediction accuracy vs. accepted overhead costs for probing

different time complexities of the considered algorithms, e.g., *DBSCAN* (cf. Table III).

In conclusion, we found that an increasing number of micro task samples improves the prediction accuracy of the regression models more and more. However, on the contrary, the total data size of the probing procedure, which contains the number of micro tasks and the data size of the particular micro tasks, is relevant in terms of overhead costs and practicality.

C. What is the Best Tradeoff Between Accuracy and Costs for Probing Unknown Computing Services?

To show the feasibility of our approach, we investigate the best tradeoff between a high prediction accuracy and the number of manageable probing samples in order that probing is still an advantage. In other words, even parallel probing of multiple offloading services should be as energy-efficient and accurate as possible to make an appropriate offloading decision and avoid offloading large actual tasks to a busy or low-performance backend system.

Figure 8 shows the results of the tradeoff analysis. The best tradeoff for each algorithm represents the data point that is closest to the top left corner of the plot. We can see that the prediction accuracy reaches an accurate level over 80% and up to 85.5% considering two micro task samples for *radix sort* (cf. Figure 8a) and *DFT* (cf. Figure 8c). To conduct these two micro-benchmarks, only $5Ws$ or $3Ws$ are required. Considering that these micro-benchmarks only take few hundred milliseconds, this is an accurate and energy-efficient way to probe backend systems. For *DBSCAN*, we get 62.4% considering two samples or over 72% considering at least three samples (cf. Figure 8b). Consequently, we recommend to preventively execute one micro task more for unstable algorithms such as *DBSCAN* with different average and worst case time complexities than for stable algorithms such as *radix sort* or *DFT*.

In summary, using the proposed probing strategy, we can accurately predict the completion times of large actual offloading tasks in an energy-efficient way only by offloading two micro tasks, which run in the range of milliseconds. This shows the feasibility and applicability of our probing approach for offloading decision support that can handle unknown third-party services requiring no prior knowledge about these offloading systems and making no assumptions for real-world deployments.

VII. Conclusion

In this paper, we proposed a novel approach that overcomes the cold-start problem for assessing and considering unknown services at runtime to make better offloading decisions. For that, our approach first probes these services by offloading micro tasks running in the range of few milliseconds for accurately predicting the performance for larger offloading tasks using regression models. We evaluated our approach on three algorithms with different time complexities. The results show that we achieved an accuracy up to 85.5% after two micro task samples for predicting the runtime performance of unknown services, which is an adequate trade-off between high prediction accuracy and low probing overhead costs. We also show that additional performance samples further improve our regression models and reduce the prediction error. Existing offloading decision algorithms can now benefit from our results and use this probing approach as supplement to consider unknown services in their decisions. We also plan to realize a generic open-source framework which is able to offload arbitrary resource-intensive tasks in a smart and fully automatic way by considering our probing approach for unknown computing services.

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the NICER project and by the German Federal Ministry of Education and Research (BMBF) Software Campus project "PersonalAssistant" [01IS12054].

References

[1] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A Survey of Mobile Phone Sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, 2010.

[2] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile Cloud Computing: A Survey," *Future Generation Computer Systems, Elsevier*, vol. 29, no. 1, pp. 84–106, 2013.

[3] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[5] C. Meurisch, A. Seeliger, B. Schmidt, I. Schweizer, F. Kaup, and M. Mühlhäuser, "Upgrading Wireless Home Routers for Enabling Large-scale Deployment of Cloudlets," in *7th Intl. Conf. on Mobile Computing, Applications, and Services (MobiCASE'15)*. Springer, 2015, pp. 12–29.

[6] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A context Sensitive Offloading Scheme for Mobile Cloud Computing Service," in *8th International Conference on Cloud Computing (CLOUD'15)*. IEEE, 2015, pp. 869–876.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer With Code Offload," in *8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*. ACM, 2010, pp. 49–62.

[8] A. N. Khan, M. M. Kiah, S. U. Khan, and S. A. Madani, "Towards Secure Mobile Cloud Computing: A survey," *Future Generation Computer Systems, Elsevier*, vol. 29, no. 5, pp. 1278–1299, 2013.

[9] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 369–392, 2014.

[10] A. Aijaz, H. Aghvami, and M. Amani, "A Survey on Mobile Data Offloading: Technical and Business Perspectives," *Wireless Communications, IEEE*, vol. 20, no. 2, pp. 104–112, 2013.

[11] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[12] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The Case for Cyber Foraging," in *10th Workshop on ACM SIGOPS European Workshop*. ACM, 2002, pp. 87–92.

[13] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[14] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The Role of Cloudlets in Hostile Environments," *Pervasive Computing, IEEE*, vol. 12, no. 4, pp. 40–49, 2013.

[15] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the Cloud to the Mobile User," in *3th Workshop on Mobile Cloud Computing and Services (MCS'12)*. ACM, 2012, pp. 29–36.

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *1st Workshop on Mobile Cloud Computing (MCC'12)*. ACM, 2012, pp. 13–16.

[17] I. Stojmenovic, "Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-to-machine Networks," in *International Telecommunication Networks and Applications Conference (ITNAC'14)*. IEEE, 2014, pp. 117–122.

[18] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *6th Conference on Computer Systems (EuroSys'11)*. ACM, 2011, pp. 301–314.

[19] Q. Xia, W. Liang, and W. Xu, "Throughput Maximization for Online Request Admissions in Mobile Cloudlets," in *38th International Conference on Local Computer Networks (LCN'13)*. IEEE, 2013, pp. 589–596.

[20] B. Patra, S. Roy, and C. Chowdhury, "A Framework for Energy Efficient and Flexible Offloading Scheme for Handheld Devices," in *9th International Conference on Advanced Networks and Telecommuncations Systems (ANTS'15)*. IEEE, 2015, pp. 1–6.

[21] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," in *15th Symposium on Principles of Distributed Computing (PODC'96)*. ACM, 1996, pp. 1–7.

[22] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "PacketCloud: An Open Platform for Elastic In-network Services," in *8th International Workshop on Mobility in the Evolving Internet Architecture (MobiArch'13)*. ACM, 2013, pp. 17–22.

[23] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling Interactive Perception Applications on Mobile Devices," in *9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*. ACM, 2011, pp. 43–56.

[24] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Capacitated Cloudlet Placements in Wireless Metropolitan Area Networks," in *40th Intl. Conf. on Local Computer Networks (LCN'15)*. IEEE, 2015, pp. 570–578.

[25] V. Pejovic and M. Musolesi, "Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 47, 2015.

[26] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading," in *31st International Conference on Computer Communications (INFOCOM'12)*. IEEE, 2012, pp. 945–953.

[27] S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, and Y. Paek, "Fast Dynamic Execution Offloading for Efficient Mobile Cloud Computing," in *11th International Conference on Pervasive Computing and Communications (PerCom'13)*. IEEE, 2013, pp. 20–28.

[28] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive Computation Offloading for Energy Conservation on Battery-powered Systems," in *13rd International Conference on Parallel and Distributed Systems (ICPADS'07)*, vol. 2. IEEE, 2007, pp. 1–8.

[29] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "COSMOS: Computation Offloading as a Service for Mobile Devices," in *15th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'14)*. ACM, 2014, pp. 287–296.

[30] H. Flores, S. N. Srirama, and R. Buyya, "Computational Offloading or Data Binding? Bridging the Cloud Infrastructure to the Proximity of the Mobile User," in *2nd Intl. Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud'14)*. IEEE, 2014, pp. 10–18.

[31] Y. Geng, W. Hu, Y. Yang, W. Gao, and G. Cao, "Energy-Efficient Computation Offloading in Cellular Networks," in *23rd Intl. Conference on Network Protocols (ICNP'15)*. IEEE, 2015, pp. 145–155.

[32] C. Meurisch, B. Schmidt, M. Scholz, I. Schweizer, and M. Mühlhäuser, "Labels - Quantified Self App for Human Activity Sensing," in *17th International Conference on Ubiquitous Computing (UbiComp'15): Adjunct Publications*. ACM, 2015, pp. 1413–1422.

[33] F. Kaup, M. Wichtlhuber, S. Rado, and D. Hausheer, "Can Multipath TCP Save Energy? A Measuring and Modeling Study of MPTCP Energy Consumption," in *40th Conference on Local Computer Networks (LCN'15)*. IEEE, 2015, pp. 442–445.

[34] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." in *2th International Conference on Knowledge, Discovery and Data Mining (KDD'96)*, vol. 96, no. 34, 1996, pp. 226–231.

[35] L. Zhang, C. Zhang, J. Liu, X. Chu, K. Xu, H. Wang, and Y. Jiang, "Power-Aware Wireless Transmission for Computation Offloading in Mobile Cloud," in *25th International Conference on Computer Communication and Networks (ICCCN'16)*. IEEE, 2016, pp. 1–9.

[36] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan, "How Close is Close Enough? Understanding the Role of Cloudlets in Supporting Display Appropriation by Mobile Users," in *10th International Conference on Pervasive Computing and Communications (PerCom'12)*. IEEE, 2012, pp. 122–127.

[37] C. Wang and Z. Li, "A Computation Offloading Scheme on Handheld Devices," *Journal of Parallel and Distributed Computing*, vol. 64, no. 6, pp. 740–746, 2004.

[38] R. J. Hyndman and A. B. Koehler, "Another Look at Measures of Forecast Accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.