Reducing Task Completion Time in Mobile Offloading Systems through Online Adaptive Local Restart

Qiushi Wang
Department of Mathematics and Computer Science
Freie Universität Berlin
Takustr.9, Berlin, Germany
qiushi.wang@fu-berlin.de

Katinka Wolter
Department of Mathematics and Computer Science
Freie Universität Berlin
Takustr.9, Berlin, Germany
katinka.wolter@fu-berlin.de

ABSTRACT

Offloading is an advanced technique to improve the performance of mobile devices. In a mobile offloading system, heavy computations are migrated from resource constrained mobile devices to powerful cloud servers through a wireless network connection. The unreliable wireless network often disturbs system operation. Task completion can be delayed or interrupted by congestion or packet loss in the network. To deal with this problem the offloaded jobs can be locally restarted and completed in the mobile device itself.

In this paper, we propose a dynamic scheme to determine whether and when to locally restart a task. First, we design an experiment to explore the impact of packet loss and delay in unreliable networks on the completion time of an offloading task. Then, we mathematically derive the prerequisites for local restart and selection of the optimal timeout. The analysis result confirms that local restart is beneficial when the distribution of task completion time has high variance. Further, a dynamic local restart scheme is proposed for mobile applications. This scheme keeps track of the variance of the probability density function of the distribution of task completion time. This is done using a dynamic histogram, which collects and updates data at run time. The efficiency of the local restart scheme is confirmed by experimental results. The experiment shows that local restart at the right time achieves better performance than always offloading.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault tolerance; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Performance, Reliability

Keywords

Mobile Offloading; Restart; Unreliable Network; Dynamic Histogram

1. INTRODUCTION

In recent years, a large number of applications have been developed for mobile devices. Obviously, many of these colourful applications have added convenience to our lives. For example, tourists will never worry about getting lost in an unfamiliar city, various navigation applications can provide the precise route information about any destination a tourist may want to visit. However, although the invention of more advanced mobile devices has improved their computational capability, the implementation of compute intensive applications is still limited by the constraint of the mobile device hardware, for example the long time operation of microchips cannot be sustained by low capacity batteries. Moreover, this constraint is not merely a temporary technological deficiency but is intrinsic to mobility [32]. The trend in development of mobile device architectures and batteries shows the difficulty to overcome this constraint in the near future. Therefore, the concept of offloading to the Cloud is employed to handle performance problems [14]. By migrating heavy computation to resourceful cloud servers, mobile devices can overcome the limitation of deficient resources. Repeated offloading can be necessary in image recognition, where an image search can be split into several section, that need repeated offloading. Another scenario where offloading can be applied is online game, for instance, playing chess with a computer opponent who is in Cloud.

The smooth offloading of computation from mobile devices to cloud servers depends on a fast and stable network connection, which guarantees seamless communication. With wireless networks such as WiFi, 3G or LTE, this seems in principle possible. Unfortunately, the quality of a network is not constant across space and time. Consequently, the execution of the offloading task may suffer from long delays or even failures in the network. In addition, using wireless connectivity demands high energy[6]. The limited battery capacity cannot support the mobile device to wait an unpredictable time for the network to recover, which may take very long.

As introduced in [44], if the offloading task needs an unknown time to migrate computation through the unreliable network connection, re-executing and completing the computations locally by the mobile device can save both time and energy. This re-execution mechanism is a type of restart. When the offloading task fails, the mobile device may retry offloading or restart the task using the resources in the local mobile device instead of those in the Cloud. In this paper we only consider local execution after a restart. The key problem behind restart is when to launch it. There clearly exists a tradeoff between the cost of local or remote retry and waiting for the offloading to succeed. In [44], a static method is proposed to find the optimal timeout when to restart locally by analysing the system performance using stochastic models. It has been shown through simulation that the optimal timeout changes when the quality of the network deteriorates. In this paper we confirm those previous simulation results by analysing experimental data and dynamically adapt the optimal restart time at run time in order to account for the variation of the network quality.

We have to solve several problems: First, the quality of the network must be assessed, second, the variation of the network quality must be monitored based on which then an estimate of the optimal restart timeout is computed. We assume that the system performance is positively correlated with the network quality, and use the task completion time as a metric to evaluate system performance. Although the energy consumption is also very important to evaluate performance, as introduced in [6, 11, 26], we are not able to easily determine energy usage and fall back to task completion time. We state that for our purposes there is a sufficiently strong correlation between energy consumption and task run time. To monitor the variation in network quality we dynamically build a histogram of the task completion time which provides a good and timely estimate of its distribution. We propose a method to periodically update the histogram.

The main contributions of this paper are 1) we experimentally confirm the impact of network quality on mobile offloading decisions, 2) we mathematically derive conditions for applying local restart and the optimal timeout based on a greedy method, and 3) we propose a dynamic online scheme to determine whether and when to launch a local restart.

The remainder of this paper is organized as follows: In Section 2 we briefly recapitulate the background on related concepts of mobile offloading, the restart algorithm and dynamic histogram generation. In Section 3 we introduce an experiment to study the impact of packet loss and delay on the task completion time. The experimental results confirm the need for local restart. Next, in Section 4 we describe the mathematical derivation of a condition which is used to determine whether and when to launch a local restart. The dynamic local restart scheme itself is introduced in Section 5. Its efficiency is illustrated using experiments. Finally, conclusions are in Section 6.

2. BACKGROUND AND RELATED WORK

Mobile offloading as a concept has been around for more than a decade. Thin clients using a remote infrastructure for compute-intensive tasks have already been seen as a method for addressing the challenges of distribution and mobility as in pervasive computing [33]. Powerful distributed systems as in Cloud computing aim at turning computing as utility into reality [8]. Recently, mobile offloading has been developed as to merge Cloud computing and mobile computing. Research in offloading methods can be divided into three main directions [14]: client-server communication, virtual machine migration and mobile agents.

We will now discuss related work in all three areas.

1. Client-server communication: communication can be supported by pre-installation of the application in both the mobile client and the server. In this case one can benefit from existing stable protocols for process communication between mobile and surrogate devices. This is the basis for the systems in [15, 4, 23, 19, 12, 20].

2. Virtual machine migration: offloading can be implemented as the migration of the complete virtual maching executing the application. The most fascinating property of this method is that no code is changed for offloading of a program. The memory image of a mobile client is copied and transferred to the destination server without interrupting or stopping any execution on the client. Although this method has clear advantages as it avoids having two versions of a program, it requires a high volume of transferred data [7, 11, 18, 34].

3. Mobile agents: Scavenger [21] introduces a framework that partitions and distributes heavy jobs to mobile surrogates in the vicinity rather than to cloud servers. Offloading to more than one surrogate is the merit of this framework.

Few of the above approaches tackle the problem of when to offload and which communication partner to choose. In [2] the authors design a Markov decision process to find the optimal aging control policies, which decide when to connect to the server and which network link to use.

All offloading systems mentioned so far may suffer under poor network condition and the application of well-designed fault-tolerance methods is in place. Restart is a simple and popular recovery scheme to mitigate network failures. It can be very effective for certain types of failures and its performance has been widely studied. Markov chain models and Laplace transforms have been developed to analyse the performance of restart for improving the expected task completion time [3, 27, 35, 22, 5]. These analyses strongly support the efficiency of restart if the best restart timeout is known. Their implementation in an online algorithm for practical application is not straight forward. A fast method based on iteration theory to identify the optimal restart time is presented in [25]. The algorithm is improved in [41, 40, 42]. It is tailored for Internet applications in [30].

The restart algorithm mentioned above relies on the probability density function *pdf* of the task completion time. In pratice, a density function is approximated by the corresponding histogram. Since the distribution of the completion time in a real-time system keeps changing with the operation, a dynamic method to adapt the histogram is required. In [13] the bucket width is adjusted when the number of samples in some buckets satisfy a given criterion. Histogram data can be stored in a structure called Q-digest which is a binary tree [36]. This allows to quickly find quantiles of the data set using a post-order traversal on the tree. In [16] the data stream is compressed by wavelet transform into a sketch. The quantile query is answered by estimating the original data with the sketch. All these methods can be used to set up the histogram for the restart algorithm. We do not evaluate the different algorithms in this paper. We use a width-fixed histogram and propose a cost-effective method to update the histogram at run time.

3. OFFLOADING OVER AN UNRELIABLE NETWORK

In order to observe and analyse the impact of an unreliable network connection on the mobile offloading system we design an experiment. Using the experiment we show that the performance changes in the system under changing network quality. We assume that the task completion time consists of the remote execution time and the data transmission time. Generally, the execution time is assumed constant for a given task and device and delays are added by data transfer. In particular, we assume that the task completion time on the mobile device and on the cloud server can be different, but both will be more or less constant for identical tasks at different times. The offloading completion time varies greatly because data transmission times are not the same. The impact of heavy load on the system is not considered in this paper. From experimental results these assumptions seem reasonable.

In the remainder of this section we first introduce our mobile offloading system and the sample application which we use here for demonstration purposes. Then we experimentally demonstrate how system performance varies over the day due to changing load in our wireless network over the day. The task completion time is described by fitting a distribution to selected subsets of the data. This shows that the variance in the task completion time distribution increases significantly for certain subsets of the data.

3.1 Experiment Configuration

Offloading can be beneficial if two conditions hold. First, the task must consist of heavy computation requirement and, second, a small amount of data must be transmitted between the mobile device and the server. An application which meets both requirements is Optical Character Recognition (OCR), but there are many more. OCR is a method to recognise the characters on a binary image with optional polygonal text regions. Generally, the recognition algorithm consists of three steps: 1) The layout of the image is analysed to find some baselines of the text region. 2) The text region is chopped into components based on the gaps in the baselines. 3) Each component is recognised as several characters by comparing its shape with a trained database. For details of OCR the interested reader is referred to [37].

All three steps of OCR require heavy computation. A series of complicated edits to the image like rotation, segmentation and comparison has to be done. Performing those tasks on the mobile device consumes a lot of energy. For the powerful remote server energy-usage is not a critical metric. In addition, most text images can be stored in small files of at most a few kilobytes. So the amount of data to transmit from the mobile device to the remote server is small. But still the time needed for the transmission depends on the quality of the network connection.

For the experiments a mobile phone (Samsung GT-S7568, Android 4.0) and a server (4 cores: Intel Xeon CPU E5649 2.53GHz) have been used. The mobile phone is placed in a dormitory room and connects to the Internet through Wifi (54Mbps provided by a local Telecom operator). The server is in the lab of the university campus and connects to the Internet through a LAN port of 100M. We have used the Linux command "traceroute" to track the route from the mobile phone to the server. Normally, the route passes 12 hops to reach the destination, and the total round-trip time is around 82ms. The offloading engine as introduced in [44] includes an Android Application (App) for the mobile client and a website project for the server. In our experiment, the Tesseract OCR Engine [1] is implemented in both parts of the offloading engine. An image (1160×391px, 8.1 KiB) with a rectangle text region, as shown in Fig. 1, is used for image recognition. Only 100 Bytes are used to represent the decyphered words.

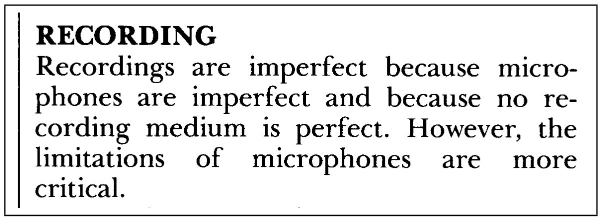
RECORDING
Recordings are imperfect because microphones are imperfect and because no recording medium is perfect. However, the limitations of microphones are more critical.

Figure 1: The image to be recognised

Completion of an offloaded OCR task can be divided into three phases: 1) the Android application transmits the image from the mobile device to the server, 2) the words on the image are recognised using the OCR engine in the server, and 3) the mobile device receives and displays the result from the server. The Offloading Completion Time(OCT) is the time needed to complete the three steps. The same offloading task has been repeated more than 58 000 times in approximately 24 hours in order to observe OCT under the different network conditions. The results are stored in a text file in the mobile device. The memory of the mobile phone used for caching is cleared after the task completion and reused again in the next new task.

In addition, we conducted a different experiment where the image recognition is performed in the mobile device. We call it local execution, as all the processing steps (e.g. analysis, chopping and recognition) are completed by the mobile device itself. The completion time is called Local Completion Time(LCT). The same image Fig. 1 is repeatedly recognised 8 400 times by the local execution. In the next subsection, we will show that although local execution is slower than offloading, it is more stable than the latter.

Fig. 2 shows a scatter plot of all data of the entire 58 000 samples over a 24-hour period starting at 8am on 14th January 2014. Under the assumption of a constant processing time, a large total completion time can be attributed to a long transmission time, i.e. poor network performance. The majority of the samples fall into the range between 980ms and 1380ms, corresponding to the 0.05 and 0.75 quantile of all the samples. Obviously the distribution of the sample values is not identical at different times. While we do not know the reason for systematic changes in network transmission times, there are clearly several types of typical behaviour that should be distinguished.

We have selected three subsets of our observations as indicated by the shaded areas in Fig. 2, each containing 2000 samples, which corresponds to a time window of 40 minutes each. The number of samples is enough to decently fit a distribution and capture one type of network behaviour, the normal, the deteriorated and the bad state.

3.2 Experimental Results

Table 1 shows the mean, the quantiles and the variance of the three subsets. In the *normal* subset the mean completion

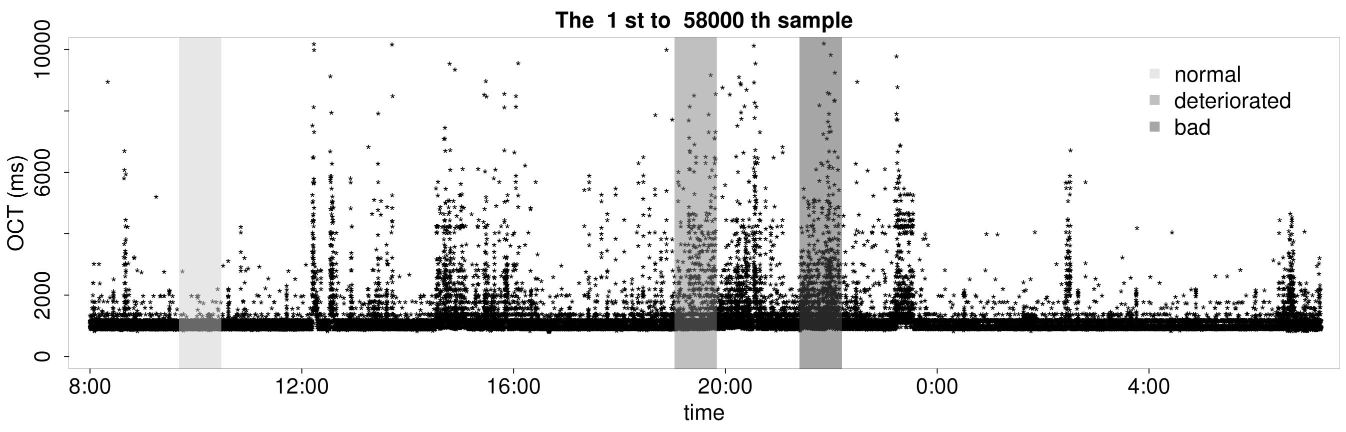

Figure 2: Scatter plot of all OCT samples

time has a low variability, as the 0.9 quantile is only 15% higher than the mean. It is also worth mentioning, that for the given application and setup fast offloading takes in total only half as long as local computation, because remote servers are much faster than mobile devices.

Table 1: Statistics of completion times (msec)

	Normal	Deteriorated	Bad	LCT
mean	1191	1618	2183	2377
0.6-quantile	1171	1466	2075	2382
0.9-quantile	1358	2595	3027	2411
0.99-quantile	1575	5495	7514	2480
variance	14496	80 5861	1680265	1249

Very roughly speaking, it seems like the network degrades most in the early afternoon and in the evening. We do not try to explain this, as finding the cause for network delays is not the scope of this paper. Rather we argue that offloading, as well as a local restart make sense for certain network condition and since we rightfully assume that network conditions change over time a sliding window estimate is needed and appropriate.

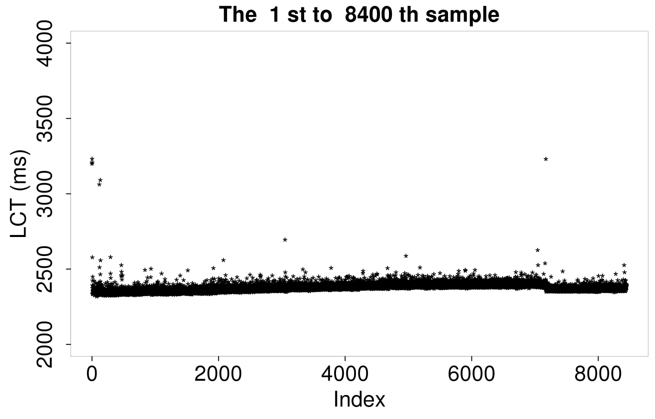

Figure 3: Scatter plot of all LCT samples

It should be noted that on the average, even in poor network condition the offloaded task completes faster than the one that is computed locally. However, for the bad network period, since enough outliers skew the distribution and increase the sample variance, the variability in the data is high enough to justify the use of restart.

The completion time measurement of the local computation (LCT) are shown in Fig. 3. Local computation is usually stable, with very few outliers. Most samples fall into a narrow range between 2338ms and 2411ms, corresponding to the 0.05 and 0.9 quantile.

In summary, in the best case offloading can provide a solution in approximately half the time needed for local processing. On the other hand, local execution times are very stable, albeit longer than processing using offloading, which suffers from high variability and, hence, sometimes takes very long.

3.3 Data Analysis

In this section the sampled data will be analysed to determine whether the theoretical conditions for successful restart are met. It can be shown [42] that restart is beneficial if the task completion time follows a distribution with sufficiently high variance or heavy-tail. Therefore, the distribution of the experimental data and its variability will be determined. The log-log complementary distribution plot is used to illustrate the weight of the tail of the distribution [10].

Fig. 4 shows the completion time of the three subsets and the local completion time versus their complementary cumulative distributions on a log scale. Clearly, for the subset of the bad network state the curve has an approximately constant slope of -2, indicating a heavy tail [10]. For the subset in deteriorated condition the tail has an exponential decay for long task completion times. Therefore in this case we cannot clearly diagnose a heavy-tailed distribution. For the normal subset the decrease is steep, for local computation completion times it is almost infinite. This indicates certainly no heavy tail in the latter two subsets.

Completion times using the local computation are almost constant. There is very little variation in the measurements. This means that once local computation has started restart will certainly not be beneficial. However, during a phase of poor network quality, a local restart may speed up the

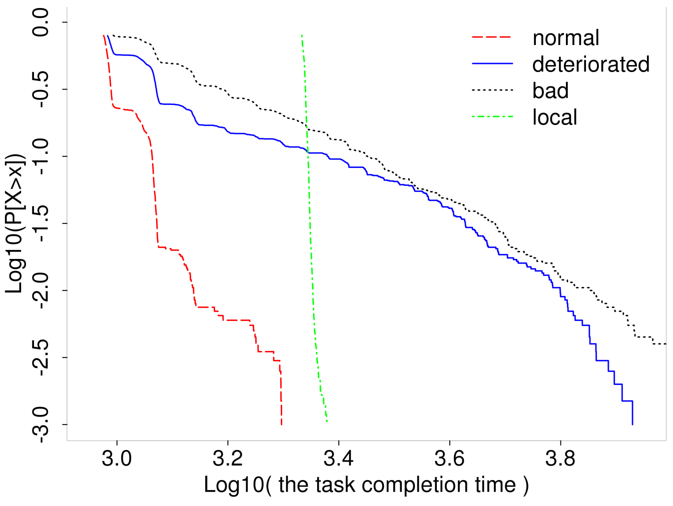

Figure 4: Log-log complementary distribution of the completion time

solution. This does not yet answer the question what a good choice of the timeout for restart could be.

Fig. 5-7 show the histograms of the three subsets and the density of the fitted distribution. For convenient fitting of phase-type distributions the histograms have been shifted to the origin by subtracting the minimum value from all observations. The distribution fitting will be discussed in the next section.

3.4 Distribution Fitting

In this section we will describe the fitting process for the offloading completion time (OCT) as shown in the histograms and densities in Figs. 5-7. Let the random variable T_o represent OCT of an offloading task without restart. The distribution of T_o is fitted with the Cluster-based fitting algorithm [29] that fits a phase-type (PH) distribution to the data. The fitting procedure uses clustering and fits an Erlang distribution to each cluster. The full distribution is then a mix of those Erlang distributions, a hyper-Erlang distribution.

The hyper-Erlang distribution is suitable for situations where restarts succeed [31]. This distribution takes values from different random variables with different probabilities, for instance, with probability α_i a value from an Erlang distribution with m_i phases and parameter $\lambda_i > 0$, $i = 1, 2, ..., M$. M is the number of clusters. In general, the mixed-Erlang distribution is represented by a vector-matrix tuple $(\boldsymbol{\alpha}, \mathbf{Q})$.

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1 & \mathbf{0} & \\ & \ddots & \ddots \\ & & \mathbf{0} \\ & & \boldsymbol{Q}_M \end{bmatrix}, \boldsymbol{Q}_i = \begin{bmatrix} -\lambda_i & \lambda_i & & \\ & \ddots & \ddots & \\ & & -\lambda_i & \lambda_i \\ & & & -\lambda_i \end{bmatrix} \quad (1)$$

$$\boldsymbol{\alpha} = (\underbrace{\alpha_1, 0, ..., 0,}_{m_1} \alpha_2, 0, ..., \underbrace{\alpha_M, 0, ..., 0,}_{m_M}) \qquad \sum_{i=1}^{M} \alpha_i = 1 \quad (2)$$

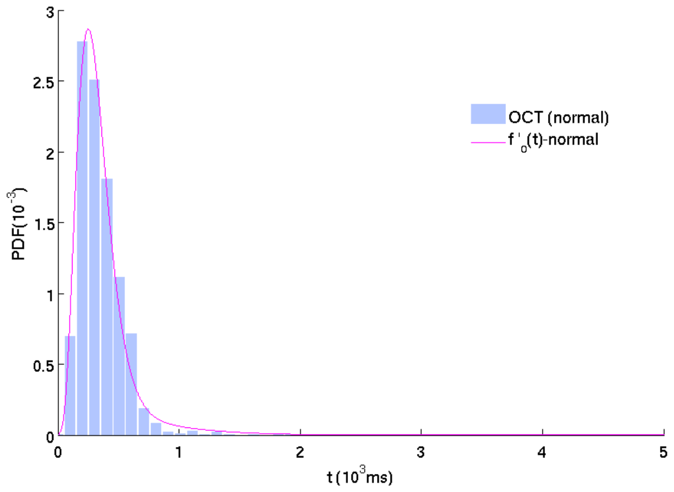

Figure 5: Histogram and PH distribution of the normal subset

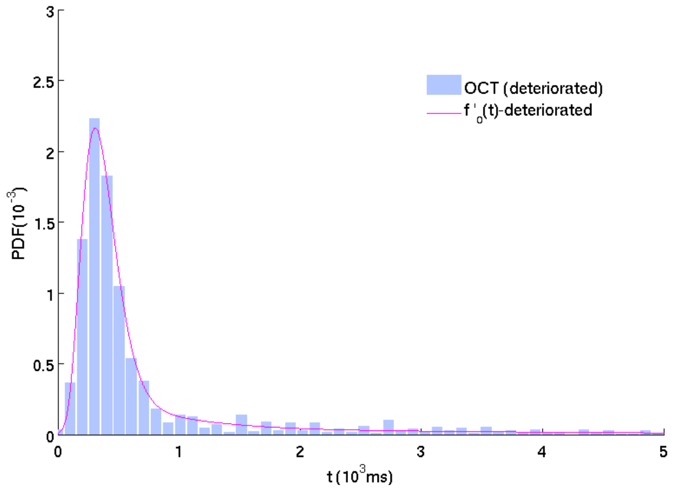

Figure 6: Histogram and PH distribution of the deteriorated subset

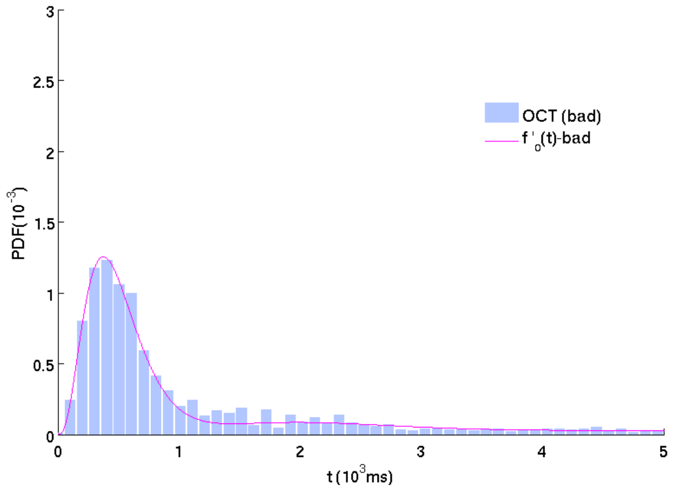

Figure 7: Histogram and PH distribution of the bad subset

$\boldsymbol{Q}_i \in \mathbb{R}^{m_i \times m_i}, i = 1, ..., M$ is a square matrix with size m_i. The probability density function and cumulative distribution function are defined as:

$$f(t) = \boldsymbol{\alpha} e^{\boldsymbol{Q}t}(-\boldsymbol{Q} \cdot \mathbf{I}) \tag{3}$$

$$F(t) = 1 - \boldsymbol{\alpha} e^{\boldsymbol{Q}t} \cdot \mathbf{I}, \tag{4}$$

where $\mathbf{I}$ is the column vector of ones with the appropriate size.

Although the hyper-Erlang distribution has exponentially decaying tails, its variance can still be large enough to fulfil the requirements for successful restart as formally introduced in Section 4.1.

Since the completion times of a task have a lower threshold greater zero, as can be seen in Fig. 2 and PH-distributions preferably have a non-zero density at the origin, we have shifted the density $f_o(t)$ to the left by the minimum observed value T^o_{min} for T_o, i.e. $f_o(t) = f'_o(t - T^o_{min})$. This yields $f'_o(t)$ as the PH fitting result of the experimental data shifted to the origin.

Table 2: Hyper-Erlang parameters

T^o_{min}	806

Phase-Type Distribution

	m	λ	α
normal	[5, 2, 3]	[0.016, 0.0041, 0.0037]	[0.88, 0.047, 0.073]
det *	[3, 6, 2]	[0.00082, 0.0163, 0.0023]	[0.1, 0.7, 0.2]
bad	[4, 8, 4]	[0.008, 0.0036, 0.001]	[0.7, 0.15, 0.15]

* det = deteriorated.

Fig. 5-7 show the histograms and the PH results of the shifted T_o of the normal, deteriorated and bad subset. We used three clusters to fit the data, $M = 3$. Since we grouped the data into three categories this seemed to be a natural choice. Of course, one could have chosen more clusters, which might have increased the accuracy of the fit. The parameter results are shown in Table 2. Table 3 shows the error measurement of the PH results of the three subsets. We use the area difference between densities $\triangle f$ and the relative error in the first moment e_1 to measure the error. $\triangle f = \int_0^\infty |\hat{f}(t) - f(t)|dt$ and $e_1 = \frac{|\hat{c_1} - c_1|}{c_1}$, $f(t)$ denotes the empirical *pdf* of the distribution to be fitted, $\hat{f}(t)$ is the *pdf* of the PH result, c_1 and $\hat{c_1}$ is the first standardized moment of the empirical distribution and of the fitted PH distribution, respectively.

Table 3: Error

	Normal	Deteriorated	Bad
$\triangle f$	0.2783	0.3051	0.2921
e_1	0.1077	0.0262	0.2894

4. OPTIMAL LOCAL RESTART

When using restart one has to decide whether and when to abort a running task and to restart it. Obviously, there is a trade-off between waiting for the offloading task to complete and terminating the attempt to try again locally. In [42], an iterative solution for an infinite number of possible retries has been derived. In this section, we adopt the solution for computing the optimal timeout from [42] for two tries and a single restart: a first attempt using offloading and a fall back local computation after expiry of the timeout. The efficiency of the method is shown in experiments. In the next section we derive an expression that formulates a condition under which restart in our offloading scenario will be beneficial. In the following section we derive the optimal timeout after which to restart.

4.1 Derivation of the Restart Condition

The theoretical concept of restart applies to random variables for which, first, two successive tries are statistically independent and identically distributed, and, second, new tries abort previous attempts. In the mobile offloading system, the second assumption is certainly met. When the mobile device restarts the task by a local try it abandons the first try on the remote server, where it might continue to run, but will not influence further processing of the restarted task. However, the two successive tries are not drawn from the same distribution, as the computation time in the local device follows a different distribution than the offloading task. The offloading timeout might not be optimal, but completion of the task is guaranteed as the local computation always finishes.

In this section the sampled data will be analysed to determine whether the theoretical conditions for successful restart are met. For a given random variable T describing task completion time restart after a timeout τ is promising if the following condition holds [42]:

$$E[T] < E[T - \tau | T > \tau] \tag{5}$$

The interpretation of condition (5) means that for restart to be beneficial the expected completion time when restarting from scratch must be less than the expected time still needed to wait for completion. It can be shown [42] that condition (5) holds if the task completion time follows a distribution with sufficiently high variance or heavy-tail.

Remember that T_o represents the offloading completion time OCT of an offloading task without restart. Its density is $f_o(t)$ and its distribution function is $F_o(t)$. Assume τ is the restart time, at which the previous offloading task is aborted and the local computation is issued. Correspondingly, T_l represents the local computation time LCT of the same task, $f_l(t)$ its density and $F_l(t)$ its distribution. We assume that $F_o(t)$ and $F_l(t)$ are both continuous probability distribution functions defined over the domain $[0, \infty)$, such that $F_o(t) > 0$ and $F_l(t) > 0$ if $t > 0$. We introduce T to denote the completion time when a local restart is allowed. We write $f(t)$ and $F(t)$ for its density and cumulative distribution function, respectively. We are interested in the expectation of T using the optimal timeout τ.

$$F(t) = \begin{cases} F_o(t) & (0 \leqslant t < \tau) \\ 1 - (1 - F_o(\tau))(1 - F_l(t - \tau)) & (\tau \leqslant t) \end{cases} \tag{6}$$

$$f(t) = \begin{cases} f_o(t) & (0 \leqslant t < \tau) \\ (1 - F_o(\tau)) f_l(t - \tau) & (\tau \leqslant t) \end{cases} \tag{7}$$

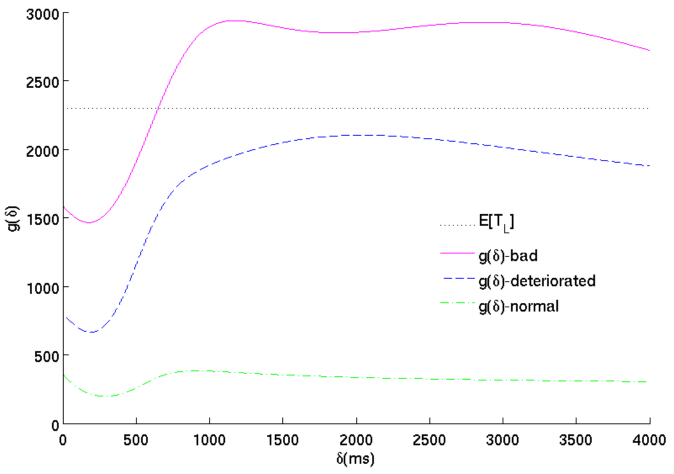

Figure 8: Restart timeout for the different subsets of the data

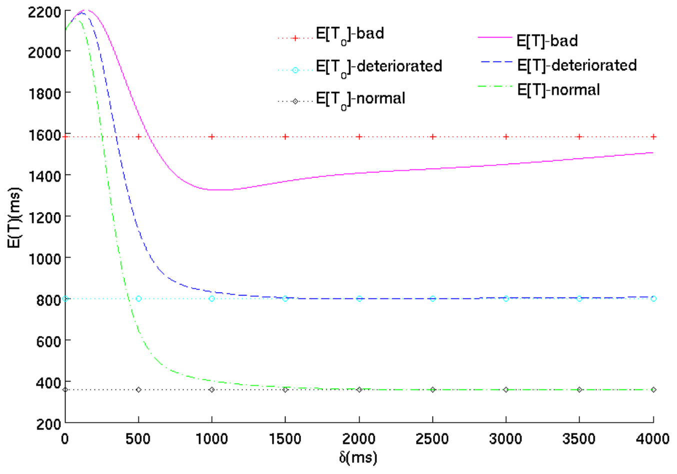

Figure 9: Expectation of OCT with/without the local restart versus τ

Analogous to [42] we define the partial moments $M_n(\tau)$ of the completion time T to determine its expectation $E[T]$.

$$M_n(\tau) = \int_0^\tau t^n f(t)dt = \int_0^\tau t^n f_o(t)dt \tag{8}$$

The respective densities of T and T_o are identical between 0 and τ, so their partial moments are equal.

$$\begin{aligned} E[T^n] &= \int_0^\tau t^n f_o(t)dt + \int_\tau^\infty t^n(1-F_o(\tau))f_l(t-\tau)dt \\ &= M_n(\tau) + (1-F_o(\tau))\sum_{k=0}^{n} \binom{n}{k} \tau^{n-k}E[T_l^k] \end{aligned} \tag{9}$$

$$E[T] = M(\tau) + (1-F_o(\tau))(\tau + E[T_l]) \tag{10}$$

A simple criterion to decide whether to restart or not can be formulated. If there exists an interval S in $[0, \infty)$, where $\tau \in S \Rightarrow E[T] < E[T_o]$, then restart is beneficial. With (10), this condition can be written as the following inequality:

$$E[T_l] < \frac{\int_\tau^\infty t f_o(t)dt}{1-F_o(\tau)} - \tau \tag{11}$$

Since the data has been shifted to the origin (11) has to be adjusted to

$$E[T_l] < \frac{\int_{\tau-T^o_{min}}^\infty t f'_o(t)dt}{1-F'_o(\tau - T^o_{min})} - (\tau - T^o_{min}) \tag{12}$$

The optimal restart time is the value of τ where $E[T]$ is minimal. Hence, $f_o(t)$ is the key factor for finding the optimal τ and to take the decision to restart. As introduced in Section 3.3, $f_o(t)$ changes with the network quality. Accurately capturing $f_o(t)$ at run time gives a good solution, but it is a challenge. In the next section, we will introduce a fast method to dynamically approximate $f_o(t)$. Before that, we use the previous experiment data to test the validity of the local restart condition (12).

4.2 The Optimal Restart Timeout

For convenience we use $g(\delta)$ to represent the right hand side of (12), $\delta = \tau - T^o_{min}$, i.e.

$$g(\delta) = \frac{\int_\delta^\infty t f'_o(t)dt}{1-F'_o(\delta)} - \delta \tag{13}$$

The potential benefit of the local restart is expressed by $g(\delta)$ and $E[T_l]$ is the threshold to decide whether the local restart is useful or not. If the value of $g(\delta)$ is low, it indicates that the task has a high probability to be completed by offloading and local restart is not helpful. If the value of $g(\delta)$ is high, it indicates that the network condition is poor and the task completion has a high probability to be delayed. In this case restart can be very beneficial to the task.

Fig. 8 shows the result of (13), calculated according to $f'_o(t)$ of the three subsets from Table 2. $E[T_l]$ is calculated based on the data in Fig. 3. Only for values δ for which $g(.)$ is larger than the expected local completion time $E[T_l]$ a retry will be beneficial. It can be seen in Fig. 8 that such values only exist for the curve based on the bad subset of data.

However, Fig. 8 does not allow to determine the optimal restart timeout. We use the expectation of T as a metric to evaluate the system performance under different restart timeouts. The optimal time is found when $E[T]$ is minimal. Equation (10) is used to calculate $E[T]$.

For comparing the system performance with and without the local restart, Fig. 9 shows $E[T]$ and $E[T_o]$ for the three subsets. As expected only $E[T]$-bad benefits from restart and even has a clear minimum under restart. The optimal restart time is found at the value for δ, for which $E[T]$-bad is minimal. We can confirm observations we already made earlier for restart, that when in doubt, one should rather set the timeout too large. A too large timeout may not be optimal, but still better than no restart. While a too small restart timeout can be detrimental to the expected task completion time. Fig. 9 confirms this observation. The figure also shows that none of the other subsets benefit from restart.

Since changes in network conditions and hence the histogram can be expected, a dynamical method is needed. In the next section, we propose a fast and simple method to dynamically update the histogram and to estimate the restart condition directly from the histogram without first fitting a distribution.

5. DYNAMIC RESTART SCHEME

The procedure of fitting a theoretical distribution and computing the optimal restart timeout from this distribution is very expensive in terms of computation cost. Various algorithms and tools exist for fitting PH distributions to empirical data [38, 17, 39, 43], and the fitted distributions approximate the data in many cases very well. For efficiency reasons we use a direct method [30] to estimate $g(\delta)$ and $E[T]$ from the histogram. We dynamically build and update a histogram and then repeatedly determine the optimal restart timeout as discussed in the following subsections.

5.1 Dynamic Histogram

A histogram simply divides up the range of possible observations into intervals, which we call buckets, and counts the number of observations that fall into each bucket. Buckets can have a variable or a constant width; we choose the latter for simplicity. Histograms initially hold too few samples to provide a good approximation of a probability distribution. After collecting data for a while a stationary distribution is represented increasingly well. However, if the distribution changes, old samples will never be dismissed from the histogram and will forever bias the new probability distribution.

There are several options how to handle changes in distribution: the histogram can be repeatedly flushed as to build up a new histogram for the respective current state of the system. This introduces many initial periods with insufficient data. Another option is to transform the buckets into *dripping buckets* that lose samples constantly over time. It is not easy to adjust the dripping speed such that the histogram will hold sufficient but not too many samples at all times [28, 24, 36].

We propose a partial flush which is tuned using two parameters, the total number of samples in the histogram when executing the partial flush and the percentage of samples to equally flush from all buckets.

Algorithm 1 (Initialization for the histogram)

```
$T_l \leftarrow$ Local_Run()    //Complete the task by local execution
$T^o_{min} \leftarrow$ Offload_Run()    //Complete the task by offloading
$T^o_{max} = T_l$
$\triangle_B = (T^o_{max} - T^o_{min})/N$    //$\triangle_B$: The bucket width
for $i = 1$ to $N$ do
    $B_{average}[i] = 0$
    $N_B[i] = 0$
end for
$N_{out} = 0$
$B_{out} = 0$
```

Algorithm 1 shows the algorithm to initialise the histogram prior to run time. The parameters are the following:

T^o_{min}: The lower bound of the histogram.

T^o_{max}: The upper bound of the histogram.

T_l: The task completion time by local execution.

N: The number of buckets in the histogram.

$B_{average}[i]$: The mean of all the samples in the ith bucket.

$N_B[i]$: The number of samples in the ith bucket.

N_{out}: The number of samples, whose value $> T^o_{max}$.

B_{out}: The mean of all the samples $> T^o_{max}$.

The number of buckets N must be chosen manually. The upper bound of the histogram is determined by the execution time of one local run. The lower bound is given as the execution time of one offloading task. In the course of the experiments there may later be shorter offloading times which will be used as new lower bound and additional buckets will be inserted. These choices are motivated by the purpose of the histogram: to determine the optimal restart timeout the precise shape of the distribution in the tail is not needed.

Algorithm 2 (Recording a new sample)

```
Local Execution:
 1: $T_{temp} \leftarrow$ Local_Run()
 2: $T_l = (T_l + T_{temp})/2$
Offloading:
 3: $T_{temp} \leftarrow$ Offload_Run()
 4: switch $T_{temp}$ do
 5:     case 1 : $T_{temp} \geqslant T^o_{max}$
 6:         $B_{out} = \frac{(B_{out} \times N_{out}) + (T_{temp} - T^o_{min})}{N_{out}+1}$
 7:         $N_{out}++$
 8:     case 2 : $T_{temp} < T^o_{min}$
 9:         $M = \lceil (T^o_{min} - T_{temp})/\triangle_B \rceil$
10:         INSERT($M$)
11:         $B_{average}[1] = T_{temp} - T^o_{min}$
12:         $N_B[1] = 1$
13:     case 3 : $T^o_{min} \leqslant T_{temp} < T^o_{max}$
14:         $j = \lfloor (T_{temp} - T^o_{min})/\triangle_B \rfloor + 1$
15:         $B_{average}[j] = \frac{(B_{average}[j] \times N_B[j]) + (T_{temp} - T^o_{min})}{N_B[j]+1}$
16:         $N_B[j]++$
17:
18: function INSERT($k$)
            //Insert $k$ empty buckets between $T_{temp}$ and $T^o_{min}$
19:     $N = N + k$
20:     $T^o_{min} = T^o_{min} - \triangle_B \times k$
21:     for $i = 1$ to $N$ do
22:         $B_{average}[i + k] = B_{average}[i] + \triangle_B \times k$
23:         $N_B[i + k] = N_B[i]$
24:     end for
25: end function
```

Algorithm 2 shows the algorithm to record a new sample at run time. If the sample comes from local execution, T_l is updated by the mean of its original value and the new sample. Hence, the impact of old samples is reduced and replaced by that of new ones.

If the new sample is produced by offloading, it can be added to the histogram in three ways according to its value. **Case 1**, when new samples are larger than T^o_{max}, they are all added to the *out* bucket. **Case 2**, when a shorter offloading time arrives, M additional buckets are inserted, M is calculated based on the ceiling function shown in line 9. T^o_{min} moves down to include the new sample. Line 21 $\sim$ 24 adjusts the mean and index of each original bucket accordingly. **Case 3**, when the sample falls into the range between T^o_{min} and T^o_{max}, it is added to the corresponding bucket in the histogram. Fig. 10 is the illustrative diagram of the three cases.

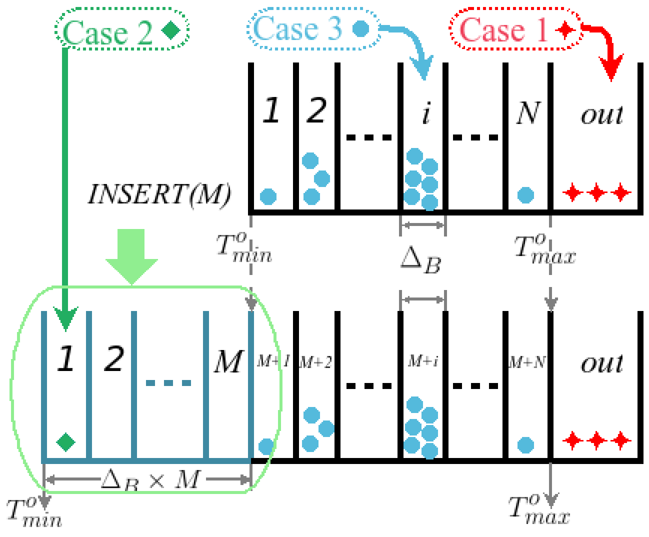

Figure 10: Recording a new offloading sample

The partial flush algorithm, shown as Algorithm 3, needs the two new parameters N_{bound} and p:

N_{bound}: threshold to start the update. When the number of samples stored in the histogram exceeds this value, the update algorithm is triggered.

p: percentage of samples to be kept. From each bucket, $(1-p)/100 * n_i$ samples are removed if the bucket holds a total of n_i samples before the partial flush.

A large number of samples N_{bound} until partial flush leads to a long sampling period. Conversely, a large percentage p indicates that the majority of the samples are kept after updating. This will lead to frequent inexpensive partial flushes. Please note that the mechanism is related to hysteresis as used in the control of queueing systems.

Algorithm 3 (Update for the histogram)

```
$B = \sum_{i=1}^{N} N_B[i] + N_{out}$
**if** $B > N_{bound}$ **then**
    $N_B[i] = \lfloor N_B[i] \times p \rfloor$    // $i$ from 1 to $N$
    $N_{out} = \lfloor N_{out} \times p \rfloor$
**end if**
```

5.2 Asymptotically Unbiased Ratio Estimator

The estimate for the optimal restart timeout is based on the asymptotically unbiased ratio estimator [9]. Using the dynamic histogram proposed in the last subsection, an estimator for $g(\delta)$ in equation (13) is:

$$\hat{g}(\delta_i) = \frac{\sum_{j=i}^{N} N_B[j] \cdot B_{average}[j] + N_{out} \cdot B_{out}}{(\sum_{k=i}^{N} N_B[k] + N_{out})(1 - \hat{F_o}'(\delta_i))} - \delta_i \quad (14)$$

We assume that the optimal timeout δ only takes on values $\delta_i = i \times \triangle_B, i = 1, 2, ..., N$. The cumulative distribution function $\hat{F_o}'(\delta_i)$ is estimated as:

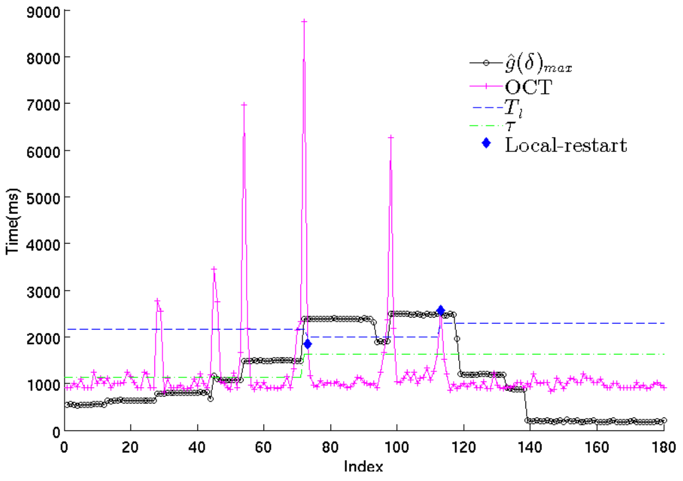

Figure 11: Scatter plot of the dynamic local restart scheme with $N = 20$, $N_{bound} = 100$ and $p = 50\%$.

$$\hat{F_o}'(\delta_i) = \frac{\sum_{j=1}^{i} N_B[j]}{\sum_{k=1}^{N} N_B[k] + N_{out}} \quad (15)$$

If the maximum estimate $\hat{g}(\delta_i)_{max} > T_l$, the local restart condition (12) is fulfilled. Then, an estimate of $E[T]$ provides the optimal timeout.

$$\hat{E}[T]_{\delta_i} = \hat{M}'(\delta_i) + (1 - \hat{F_o}'(\delta_i))(\delta_i + T_l) + T^o_{min} \quad (16)$$

Remember that we have shifted all data, and the histogram to the origin. Therefore the lower bound T^o_{min} of the histogram should be added to the expectation. The partial moment $\hat{M}'(\delta_i)$ is estimated as:

$$\hat{M}'(\delta_i) = \frac{\sum_{j=1}^{i} N_B[j] \cdot B_{average}[j]}{\sum_{k=1}^{i} N_B[k]} \quad (17)$$

The optimal local restart time can be identified by selecting the value of δ_i, which minimizes $\hat{E}[T]_{\delta_i}$, and the optimal timeout is $\tau = \delta_i + T^o_{min}$. Actually, at run time first the restart condition is evaluated and if it is not satisfied, $\hat{E}[T]_{\delta_i}$ is not determined.

5.3 Evaluation of the Dynamic Restart

In order to evaluate the performance of the dynamic local restart scheme, it is implemented in our mobile offloading engine [44] and evaluated using the OCR application with the same picture as before (cf. Fig. 1). As introduced in Section 3.2, we again conduct measurements over a period of 24 hours from 8:00 on 28th April 2014 and we sampled 54 318 completion times. Using the experiment we then show that our dynamic histogram captures changes in the system and allows the offloading system to react to those in real-time.

Fig. 11 shows a short episode of the whole experiment process. This episode lasts for about 5 minutes (begins at 9:12) and contains 180 successive tasks. A scatter plot of some related parameters of the 180 tasks is shown in Fig. 11. It can be seen that the potential benefit of the local restart, $\hat{g}(\delta)_{max}$, first increases stepwise and then remains constant. After some very long offloading times, $\hat{g}(\delta)_{max} > T_L$, several restarts complete the computation locally.

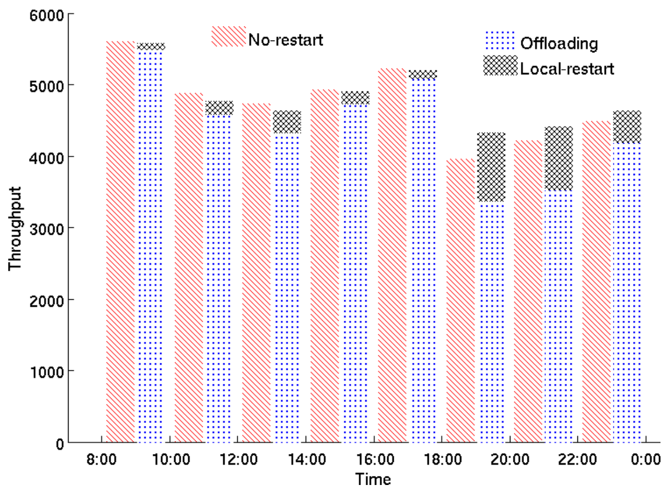

Figure 12: Throughput of different times in a day

For comparing the performance of the scheme with and without the dynamic local restart, the throughput of the two schemes over periods of two hours are shown in Fig. 12. We define the throughput as the number of tasks completed in each period. Here we compare data from the two experiment sessions that took place on different days: the right column in each interval represents the first series of experiments without restart, while the left column shows the new series of experiments using restart. Surprisingly, both columns follow a similar pattern over the day and in most intervals the throughput is almost identical in both experiment series. Only for the last three pairs of columns the dynamic local restart scheme can effectively increase the throughput.

In conclusion, the dynamic local restart scheme can effectively increase the system performance sometimes and does not harm it at any time.

6. CONCLUSION

In this paper, we have introduced a dynamic local restart scheme to improve the performance of the mobile offloading system. Restarting the offloading task again locally in the mobile device at the appropriate moment can reduce its completion time in some cases. First, we introduced an experiment to illustrate the impact of network delays on mobile offloading. Then, we mathematically derived a condition and the optimal timeout for local restart in order to reduce the task completion time. We proposed a dynamic local restart scheme for the mobile offloading system. In this scheme, a dynamic histogram is used to track the variation of the network quality, and the restart condition and the optimal time is estimated with the histogram. Since the normal user might not perform the same task many times in sequence, we have to adjust the method to suit various applications and different tasks. This might be possible by considering more fine-grained metrics such as packet transmission time to base the restart decision on.

7. REFERENCES

[1] tesseract-ocr. `http://code.google.com/p/tesseract-ocr/`.

[2] Altman, E., El-Azouzi, R., Menasche, D. S., and Xu, Y. Forever young: Aging control for smartphones in hybrid networks. *arXiv preprint arXiv:1009.4733* (2010).

[3] Asmussen, S., Fiorini, P., Lipsky, L., Rolski, T., and Sheahan, R. Asymptotic behavior of total times for jobs that must start over if a failure occurs. *Mathematics of Operations Research 33*, 4 (2008), 932–944.

[4] Balan, R. K., Satyanarayanan, M., Park, S. Y., and Okoshi, T. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services* (2003), ACM, pp. 273–286.

[5] Bobbio, A., and Trivedi, K. S. Computation of the distribution of the completion time when the work requirement is a ph random variable This work was supported in part by the US Office of Naval Research under Contract no. N3014-88-K-0623, by NASA under Grant NAG-1-70, and by the Italian National Research Council CNR under the project "Material and Devices for Solid State Electronics" Grant no. 86.02177. 61. *Stochastic Models 6*, 1 (1990), 133–150.

[6] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (2010), USENIX Association, pp. 21–21.

[7] Chun, B., Ihm, S., Maniatis, P., Naik, M., and Patti, A. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems* (2011), pp. 301–314.

[8] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (2005), USENIX Association, pp. 273–286.

[9] Cochran, W. G. *Sampling techniques*. John Wiley & Sons, 2007.

[10] Crovella, M. E., Taqqu, M. S., and Bestavros, A. Heavy-tailed probability distributions in the World Wide Web. *A practical guide to heavy tails 1* (1998), 3–26.

[11] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010), ACM, pp. 49–62.

[12] Deboosere, L., Simoens, P., De Wachter, J., Vankeirsbilck, B., De Turck, F., Dhoedt, B., and Demeester, P. Grid design for mobile thin client computing. *Future Generation Computer Systems 27*, 6 (2011), 681–693.

[13] Donjerkovic, D., Ioannidis, Y. E., and Ramakrishnan, R. Dynamic histograms: Capturing evolving data sets. In *Proceedings of the International Conference on Data Engineering* (2000), IEEE Computer Society Press; 1998, pp. 86–86.

[14] Fernando, N., Loke, S. W., and Rahayu, W. Mobile cloud computing: A survey. *Future Generation Computer Systems 29*, 1 (2013), 84–106.

[15] Flinn, J., Park, S., and Satyanarayanan, M. Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on* (2002), IEEE, pp. 217–226.

[16] Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB* (2001), vol. 1, pp. 79–88.

[17] Horváth, A., and Telek, M. Phfit: A general phase-type fitting tool. In *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 2002, pp. 82–91.

[18] Huang, D., Zhang, X., Kang, M., and Luo, J. MobiCloud: building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on* (2010), IEEE, pp. 27–34.

[19] Huerta-Canepa, G., and Lee, D. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond* (2010), ACM, p. 6.

[20] Kemp, R., Palmer, N., Kielmann, T., and Bal, H. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*. Springer, 2012, pp. 59–79.

[21] Kristensen, M. D. Scavenger: Transparent development of efficient cyber foraging applications. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on* (2010), IEEE, pp. 217–226.

[22] Kulkarni, V. G., Nicola, V. F., and Trivedi, K. S. On modelling the performance and reliability of multimode computer systems. *Journal of Systems and Software 6*, 1 (1986), 175–182.

[23] Marinelli, E. E. Hyrax: cloud computing on mobile devices using MapReduce. Tech. rep., DTIC Document, 2009.

[24] Matias, Y., Vitter, J. S., and Wang, M. Dynamic Maintenance of Wavelet-Based Histograms. In *Proceedings of the 26th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 2000), VLDB '00, Morgan Kaufmann Publishers Inc., pp. 101–110.

[25] Maurer, S. M., and Huberman, B. A. Restart strategies and Internet congestion. *Journal of Economic Dynamics and Control 25*, 3 (2001), 641–654.

[26] Miettinen, A. P., and Nurminen, J. K. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010), USENIX Association, pp. 4–4.

[27] NICOLA, V., and Trivedi, K. The completion time of a job on multimode systems. *Advances in Applied Probability 19*, 4 (1987), 932–954.

[28] Poosala, V., Haas, P. J., Ioannidis, Y. E., and Shekita, E. J. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD Record 25*, 2 (1996), 294–305.

[29] Reinecke, P., Krauss, T., and Wolter, K. Cluster-based fitting of phase-type distributions to empirical data. *Computers & Mathematics with Applications 64*, 12 (2012), 3840–3851.

[30] Reinecke, P., Van Moorsel, A., and Wolter, K. A measurement study of the interplay between application level restart and transport protocol. In *Service Availability*. Springer, 2005, pp. 86–100.

[31] Ruan, Y., Horvitz, E., and Kautz, H. Restart policies with dependence among runs: A dynamic programming approach. In *Principles and Practice of Constraint Programming-CP 2002* (2002), Springer, pp. 573–586.

[32] Satyanarayanan, M. Mobile computing. *Computer 26*, 9 (1993), 81–82.

[33] Satyanarayanan, M. Pervasive computing: Vision and challenges. *Personal Communications, IEEE 8*, 4 (2001), 10–17.

[34] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (2009), 14–23.

[35] Sheahan, R., Lipsky, L., Fiorini, P. M., and Asmussen, S. On the completion time distribution for tasks that must restart from the beginning if a failure occurs. *ACM SIGMETRICS Performance Evaluation Review 34*, 3 (2006), 24–26.

[36] Shrivastava, N., Buragohain, C., Agrawal, D., and Suri, S. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), ACM, pp. 239–249.

[37] Smith, R. An Overview of the Tesseract OCR Engine. In *ICDAR* (2007), vol. 7, pp. 629–633.

[38] Telek, M., and Heindl, A. Matching moments for acyclic discrete and continuous phase-type distributions of second order.

[39] Thummler, A., Buchholz, P., and Telek, M. A novel approach for phase-type fitting with the EM algorithm. *Dependable and Secure Computing, IEEE Transactions on 3*, 3 (2006), 245–258.

[40] Van Moorsel, A. P., and Wolter, K. Analysis and algorithms for restart. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the* (2004), IEEE, pp. 195–204.

[41] van Moorsel, A. P., and Wolter, K. Meeting Deadlines through Restart. In *MMB* (2004), pp. 155–160.

[42] Van Moorsel, A. P., and Wolter, K. Analysis of restart mechanisms in software systems. *Software Engineering, IEEE Transactions on 32*, 8 (2006), 547–558.

[43] Wang, J., Liu, J., and She, C. Segment-based adaptive hyper-Erlang model for long-tailed network traffic approximation. *The Journal of Supercomputing 45*, 3 (2008), 296–312.

[44] Wang, Q., Griera Jorba, M., Ripoll, J. M., and Wolter, K. Analysis of local re-execution in mobile offloading system. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on* (2013), IEEE, pp. 31–40.