Softw Syst Model (2018) 17:397–413
https://doi.org/10.1007/s10270-016-0531-3

THEME SECTION PAPER

Accelerating task completion in mobile offloading systems through adaptive restart

Qiushi Wang[1] · **Katinka Wolter**[1]

Received: 29 June 2015 / Revised: 15 April 2016 / Accepted: 26 April 2016 / Published online: 11 May 2016

Abstract Mobile application offloading is an efficient technique to unload the burden of intensive computation from thin clients to powerful servers. In a mobile offloading system, cloud computing is utilized to complete some heavy tasks which are migrated from resource-constrained mobile devices to the Cloud. To assure system performance, the quality of the wireless network connection plays an important role. In previous work we experimentally explored the impact of packet loss and delay in wireless networks on the completion time of an offloading task. We investigated a local restart mechanism to mitigate these effects. In the presence of unreliable communication, once the waiting time for the response of a cloud server exceeds a given threshold, exploiting the local resources of a mobile client can accelerate the task completion.

In this paper, we upgrade the restart mechanism by allowing several offloading retries before a job eventually is locally restarted and finally completed in the client device itself. This is an adaptive restart scheme which aims first at completing the job using restart with offloading. If several successive offloading attempts fail the job is completed locally. Adaptively selecting the right retry threshold and automatically restarting at the appropriate moment can balance out undesired effects. This paper extends Wang and Wolter (Proceedings of the 6th ACM/SPEC international conference on performance engineering. ACM, pp 3–13, 2015) by adding an adaptive retry scheme, a mathematical derivation of the optimal limit for offloading attempts so as to minimize the task completion time using a greedy method, and by the results of a practical evaluation study which shows the efficiency and benefits of the adaptive restart scheme.

Keywords Mobile offloading · Restart · Unreliable network

Communicated by Dr. Kai Sachs and Catalina Llado.

✉ Qiushi Wang
qiushi.wang@fu-berlin.de

[1] Department of Mathematics and Computer Science, Freie Universität Berlin, Takustr. 9, Berlin, Germany

1 Introduction

In computer and communication systems, when a task is subject to failures or unpredictable delays, aborting the previous try and launching a new process can speed up the task completion. This mechanism is generally called restart and is widely used for fault tolerance. Restart is a simple yet efficient solution which can be applied even if the probability distribution of a task completion time exhibits high variance. Examples of such tasks include randomized search algorithms, distributed data queries and data transmission through unreliable network connections. As modern computer systems become more interconnected and diverse, a large number of management work have to be completed autonomously by the system itself. Thus, these tasks are also widely spread in autonomic computing [18,19]. For instance, in order to monitor the system state, the collected system data should firstly be transmitted to some management agents. The transmission time could be high variance in a network with intermittent connectivity. Then the data have to be queried in a large historical database and mapped to some state. Once the state releases some signals that the system meets problems, a searching process is launched to find the solution to deal with this problem. Both the query and search work should be completed within a given time, but they may also suffer a high variance. As introduced in [43], restart allows to express an efficient trade-off between the average and the variance in the time a task will take.

By migrating heavy computation to powerful cloud servers, mobile offloading systems can circumvent constraints of the client hardware, e.g. high-energy consumption microchips and low capacity batteries. But smooth offloading of computation depends on a fast and stable network connection, which guarantees seamless communication. Unfortunately, the quality of a network is often not constant across space and time. In [46] the impact of unreliable network connections on mobile offloading has been experimentally confirmed. The execution of offloading tasks may suffer from long delays or even sometimes from failures. When the offloading task fails, the client may retry offloading. Various events can cause delays, for example a single node overload on the path between the mobile device and the server. The offloading data can be blocked or even lost at this node. In this situation, repeated offloading restart cannot solve the problem, and on the contrary, it increases congestion. To avoid this dilemma, the task can be completed using the resources in the local device instead of those in the Cloud. As introduced in [44], simulations of a stochastic model showed that if the offloading task needs an unknown time to migrate computation through the unreliable network connection, restarting and completing the computations locally by the mobile device can save both time and energy.

This paper is an extended version of our published at ICPE-2015 [46]. For completeness of presentation the experiment about exploring the impact of unreliable network connectivity in [46] is retained. In [46], only local restart was introduced. In this paper, both the offloading and local restart are jointly used to improve the system efficiency. This new scheme is called adaptive restart. In the new scheme, the first job attempts to restart with offloading. If the number of restarts exceeds a threshold, the job is completed locally. The main problem that must be solved is setting an optimal threshold that limits the number of allowed restart tries.

We use the task completion time as a metric to evaluate system performance under different thresholds. Besides the number of tries also the timeout is essential. The timeout controls the moment at which restart is launched. We theoretically identify the optimal threshold and timeout and experimentally confirm their advantages. Since both threshold and timeout depend on the network and system condition, setting their values is an adaptation process.

The remainder of this paper is organized as follows: in Sect. 2 we briefly review the background of mobile offloading systems and the restart algorithm. In Sect. 3 we introduce an experiment to study the impact of packet loss and delay in network on the offloading task completion time. The experimental results confirm the need for applying restart. Next, in Sect. 4 the workflow of the adaptive restart scheme is introduced. In Sect. 5 we describe the mathematical derivation of a condition which is used to determine the optimal threshold and timeout. The dynamic adaptive restart scheme implemented in a practical offloading platform is introduced in Sect. 6. The experiments and results introduced in Sect. 7 confirm the efficiency of the adaptive restart scheme. Section 8 concludes the paper.

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 [34]. Powerful distributed systems as in Cloud computing aim at turning computing as utility into reality [5]. 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 [13]: 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 [2,11,14,21,23].
2. Virtual machine migration: offloading can be implemented as the migration of the complete virtual machine 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 [4,6,36].
3. Mobile agents: Scavenger [22] 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.

All offloading systems mentioned so far may suffer from poor network condition, and the application of well-designed fault-tolerance methods is in place. Restart is a popular preventive maintenance method to mitigate network failures. It can be very effective for certain types of failures, and its performance has been widely studied [1,12,15]. Markov chain models and Laplace transforms have been developed to analyse the performance of restart for improving the expected task completion time [1,3,20,28,37]. 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 [24]. The algorithm is improved in [41–43]. It is tailored for Internet applications in [33].

Based on the assumption that the service time are phase-type distributed, Fourneau et al. [15] has proposed an open queueing network based on G-network formalism to study the effectiveness of the restart method in a multiple clients system. By comparing the analytical results and simulation results, the authors indicated that in more realistic scenarios, the arrival process of restart signals is not independent of the job arrival process. In [10], a simulation-based framework SFERA is introduced for the investigation of restart algorithms and their impact on request completion times in service-oriented system. SFERA observed systems from the client point of view and evaluated the time behaviour of a service-oriented architecture system.

Restart has also been widely deployed in high-performance computing (HPC) system. Generally, it collaborates together with the checkpoint scheme [12,16]. In [26], a checkpoint–restart framework, BlobCR is proposed specifically for the optimization of HPC applications that need to be ported to clouds. As an increasing rate of soft error is shown in recent trends, Ni et al. [27] introduced an automatic checkpoint–restart framework ACR to detect and recover the system from soft or hard faults with minimal application intervention. In [9], the authors provided a formula to compute the optimal number of checkpoint for cloud jobs under varied distributions of failure events. An adaptive algorithm is designed to optimize the impact of checkpoint–restart scheme regarding various costs like overhead.

In addition, restart is exploited in pure mathematics. A heuristic adaptive restart technique is proposed in [29] to improve the convergence rate of accelerated gradient schemes. These schemes could reduce the complexity of first-order algorithms applied to minimize smooth convex function. As the iterates generated by an accelerated gradient scheme exhibit a periodic behaviour, the optimal restart interval is proportional to this period.

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 (OCT) varies greatly because data transmission times are not the same at all times. The impact of heavy load on the system is not considered in this paper. Overload conditions may remain in the system for longer duration and show greater correlation in the task completion times. This effect is neglected by not considering overload. But for our analysis delays in the server cannot be distinguished from delays caused by the network and we only consider the latter.

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 recognize 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 recognized as several characters by comparing its shape with a trained database. For details of OCR, the interested reader is referred to [38].

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 2.53 GHz) have been used. The mobile phone is placed in a dormitory room and connects to the Internet through Wifi (54 Mbps provided by a local Telecom operator). The server

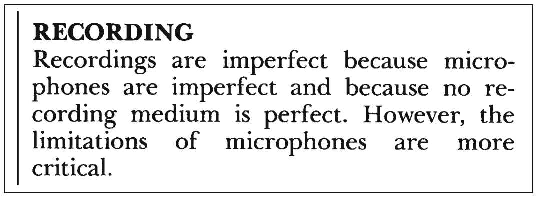

RECORDING
Recordings are imperfect because microphones are imperfect and because no recording medium is perfect. However, the limitations of microphones are more critical.

Fig. 1 Image to be recognized

is in the laboratory of the university campus and connects to the Internet through a LAN port of 100 M. 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 82 ms. 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 is implemented in both parts of the offloading engine. An image (1160 × 391 px, 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.

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 recognized using the OCR engine in the server, and (3) the mobile device receives and displays the result from the server. The 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 h 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 recognized 8400 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.

Figure 2 shows a scatter plot of all data of the entire 58,000 samples over a 24-h period starting at 8am on 14 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 980 and 1380 ms, 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 min 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 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 set-up fast offloading takes in total only half as long as local computation, because remote servers are much faster than mobile devices.

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 of network delays is not within the scope of this paper. Rather we argue that offloading as well as a local restart make sense for certain network

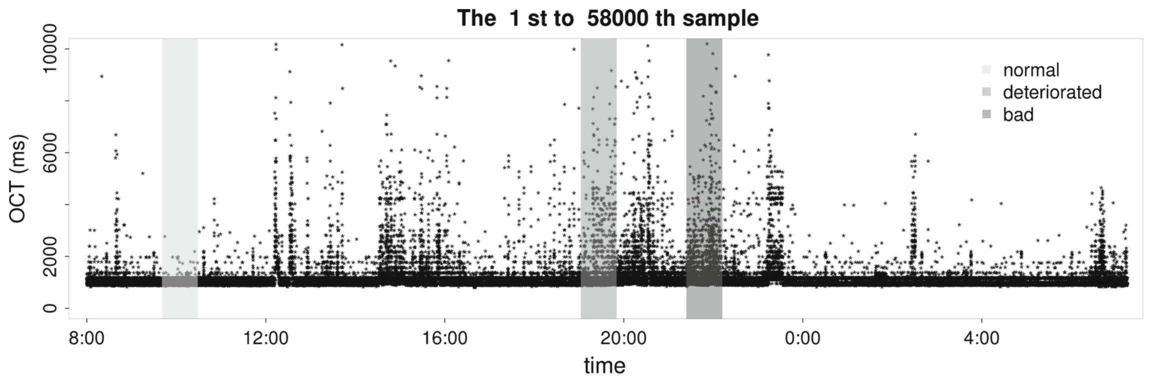


Fig. 2 Scatter plot of all OCT samples

Table 1 Statistics of completion times (ms)

	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	14,496	805,861	1,680,265	1249

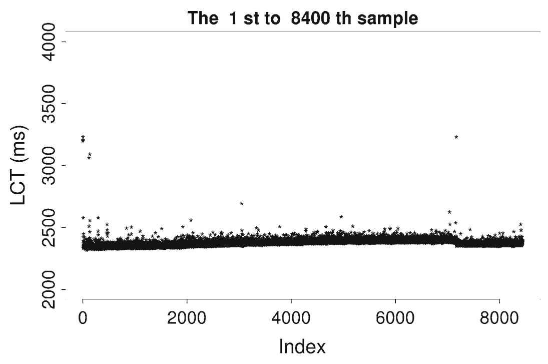


Fig. 3 Scatter plot of all LCT samples

conditions and since we rightfully assume that network conditions change over time a sliding window estimate is needed and appropriate.

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) is shown in Fig. 3. Local computation is usually stable, with very few outliers. Most samples fall into a narrow range between 2338 and 2411 ms, 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 [43] 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 [8].

Figure 4 shows the completion time of the three subsets and the LCT 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 [8]. 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, and 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 solution. This does not yet answer the question what a good choice of the timeout for restart could be.

Figures 5, 6 and 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 OCT as shown in the histograms and densities in Figs. 5, 6 and 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 [32] that fits a phase-type (PH) distribution to the data. The fitting procedure uses clustering and fits an

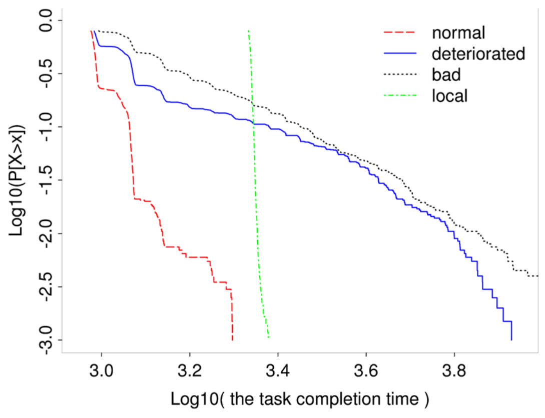


Fig. 4 Log–log complementary distribution of the completion time

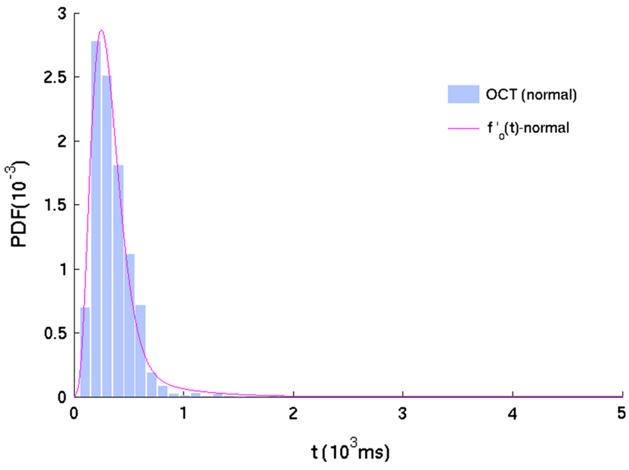


Fig. 5 Histogram and PH distribution of the normal subset

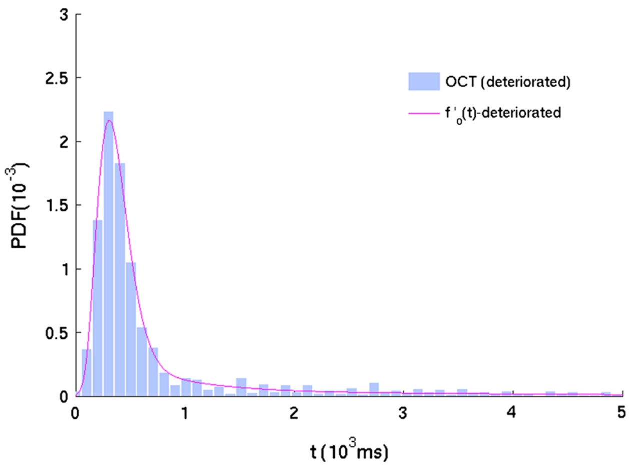


Fig. 6 Histogram and PH distribution of the deteriorated subset

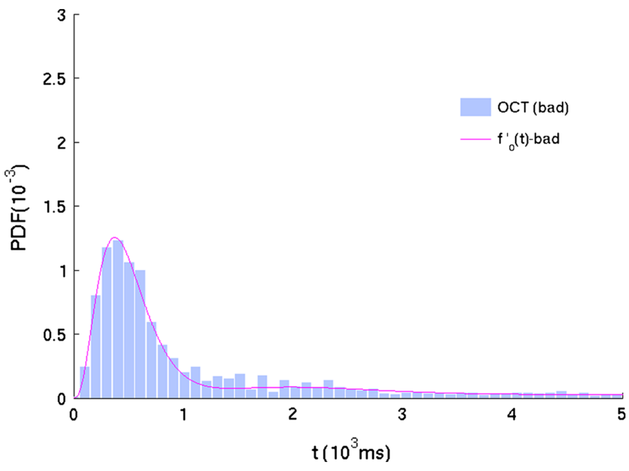


Fig. 7 Histogram and PH distribution of the bad subset

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 $\lambda_i > 0$, $i = 1, 2, \ldots, M$. M is the number of clusters. In general, the mixed-Erlang distribution is represented by a vector-matrix tuple $(\boldsymbol{\alpha}, \mathbf{Q})$.

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

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

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

$$f(t) = \boldsymbol{\alpha} e^{\boldsymbol{Q}t}(-\boldsymbol{Q} \cdot \mathbf{1}) = \sum_{i=1}^{M} \alpha_i \frac{(m_i\lambda_i)^{m_i} t^{m_i-1}}{(m_i-1)!} e^{-m_i\lambda_i t} \quad (t \geqslant 0) \tag{3}$$

$$F(t) = 1 - \boldsymbol{\alpha} e^{\boldsymbol{Q}t} \cdot \mathbf{1} = 1 - \sum_{i=1}^{M} \alpha_i \left(\sum_{k=0}^{m_i-1} \frac{(m_i\lambda_i t)^k}{k!} e^{-m_i\lambda_i t} \right), \tag{4}$$

where $\mathbf{1}$ 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 Sect. 5.1.

Since the completion times of a task have a lower threshold greater than zero, as can be seen in Fig. 2 and PH distributions preferably have a nonzero 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.

Figures 5, 6 and 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

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

Table 3 Error

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

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. $\triangle f = \int_0^\infty |\hat{f}(t) - f(t)|\mathrm{d}t$ and $e_1 = \frac{|\hat{c_1} - c_1|}{c_1}$, where $f(t)$ 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.

4 Failure handling with restart

Our concept of mobile offloading is to provide both options, i.e. of executing a given application locally in a mobile device as well as remotely in a server and selecting the suitable one of the two. In the normal system state, the mobile device can establish a reliable connection with the server to send parameters and receive results. When the connection is interrupted or suffers degradation, the system is in some kind of a failed state. The failure handling consists of restart after expiry of a timeout. The workflow of normal offloading and restart is shown in Fig. 8.

4.1 Offloading retry

We assume that during the offloading process, if the network connection is disturbed and cannot support the data transmission, the mobile device is informed to wait. A timeout value is set to restrict the waiting time. If the network recovers quickly before the timeout expires, the offloading process resumes execution. However, if the network is not robust enough, the connection is unable to recover in a short time.

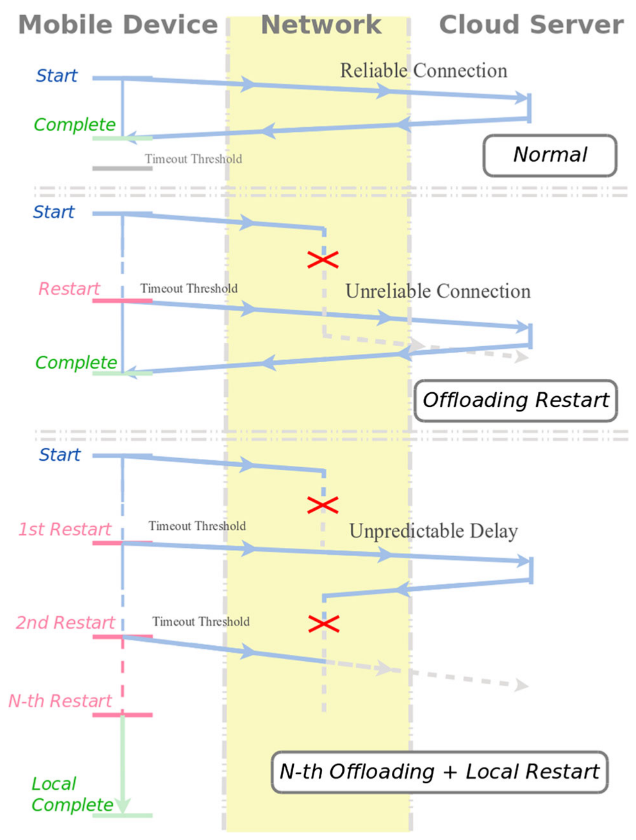


Fig. 8 Process of the offloading execution with adaptive restart

This is assumed to happen when the waiting time exceeds the timeout. In this case the previous task is abandoned, and a new try of the same offloading task is restarted.

4.2 Adaptive restart

At times the offloading process may meet an unpredictable delay in data transmission. If the connection quality experiences a long-term turbulence, the mobile device has to restart several times and waits long for a sufficiently long up-time of the connection to complete the data transmission. Obviously, the redundant restart consumes not only time but also too much energy. In order to avoid such a long waiting time, the pre-offloading task will be restarted locally in the mobile device. Although the local restart may take longer than offloading, the execution continuity is maintained.

5 Optimal adaptive 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 [43], 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 [43] for the expected task completion time of the adaptive restart scheme. Based on the theoretical analysis, the optimal timeout for every restart and the optimal threshold for the number of offloading restarts are derived.

For a given random variable T describing task completion time restart after a timeout τ is promising if the following condition holds [43]:

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

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 [43] that condition (5) holds if the task completion time follows a distribution with sufficiently high variance or heavy tail.

5.1 Derivation of the restart condition

Remember that T_o represents the 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 new try 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 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^*. The value of τ can be changed in real time according to system performance. But for simplifying the theoretical analysis, we assume that τ is constant in the process of completing a given task. Then, in the next subsection, we use individual timeout values for the restart with offloading and local execution.

$$F(t)=\begin{cases}F_o(t) & 0\leqslant t<\tau\\ 1-(1-F_o(\tau))(1-F_o(t-\tau)) & \tau\leqslant t<2\tau\\ \quad\vdots & \\ 1-(1-F_o(\tau))^{n-1}(1-F_o(t-(n-1)\tau)) & (n-1)\tau\leqslant t<n\tau\\ 1-(1-F_o(\tau))^{n}(1-F_l(t-n\tau)) & n\tau\leqslant t\end{cases} \tag{6}$$

$$f(t)=\begin{cases}f_o(t) & 0\leqslant t<\tau\\ (1-F_o(\tau))f_o(t-\tau) & \tau\leqslant t<2\tau\\ \quad\vdots & \\ (1-F_o(\tau))^{n-1}f_o(t-(n-1)\tau) & (n-1)\tau\leqslant t<n\tau\\ (1-F_o(\tau))^{n}f_l(t-n\tau) & n\tau\leqslant t\end{cases} \tag{7}$$

Remember in Eqs. (6) and (7), that $n\geqslant 2$, when $n=1$ is the single local restart mode which has been analysed in [46] as:

$$F(t)=\begin{cases}F_o(t) & (0\leqslant t<\tau)\\ 1-(1-F_o(\tau))(1-F_l(t-\tau)) & (\tau\leqslant t)\end{cases} \tag{8}$$

$$f(t)=\begin{cases}f_o(t) & (0\leqslant t<\tau)\\ (1-F_o(\tau))f_l(t-\tau) & (\tau\leqslant t)\end{cases} \tag{9}$$

We define the partial expectation $M(\tau)$ of the completion time T to determine its expectation $E[T]$.

$$M(\tau)=\int_0^\tau tf(t)\mathrm{d}t=\int_0^\tau tf_o(t)\mathrm{d}t \tag{10}$$

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

$$\begin{aligned}E[T]&=\int_0^\tau tf_o(t)\mathrm{d}t+(1-F_o(\tau))\int_\tau^{2\tau}tf_o(t-\tau)\mathrm{d}t+\cdots\\&\quad+(1-F_o(\tau))^{n-1}\int_{(n-1)\tau}^{n\tau}tf_o(t-(n-1)\tau)\mathrm{d}t\\&\quad+(1-F_o(\tau))^{n}\int_{n\tau}^{\infty}tf_l(t-n\tau)\mathrm{d}t\\&=M(\tau)+(1-F_o)(\tau)(M(\tau)+\tau F_o(\tau))+\cdots\\&\quad+(1-F_o(\tau))^{n-1}(M(\tau)+(n-1)\tau F_o(\tau))\\&\quad+(1-F_o(\tau))^{n}(n\tau+E[T_l])\\&=\sum_{k=0}^{n-1}(1-F_o(\tau))^k(M(\tau)+k\tau F_o(\tau))\\&\quad+(1-F_o(\tau))^{n}(n\tau+E[T_l]),\end{aligned} \tag{11}$$

$\because$

$$\begin{aligned}&(1-F_o(\tau))^n n\tau+(1-F_o(\tau))^{n-1}(n-1)\tau F_o(\tau)\\&\quad=(1-F_o(\tau))^n\tau+(1-F_o(\tau))^n(n-1)\tau\\&\qquad+(1-F_o(\tau))^{n-1}\tau F_o(\tau)\\&\quad=(1-F_o(\tau))^n\tau+(1-F_o(\tau))^{n-1}(n-1)\tau,\end{aligned} \tag{12}$$

$\therefore$

$$\begin{aligned}&\sum_{k=0}^{n-1}(1-F_o(\tau))^k k\tau F_o(\tau)+(1-F_o(\tau))^n\tau\\&\quad=\tau\sum_{k=1}^{n}(1-F_o(\tau))^k.\end{aligned} \tag{13}$$

Substituting (13) in Eq. (11), we get

$$\begin{aligned}E[T]&=M(\tau)+\sum_{k=1}^{n-1}(1-F_o(\tau))^k(M(\tau)+\tau)\\&\quad+(1-F_o(\tau))^n(\tau+E[T_l]).\end{aligned} \tag{14}$$

Let $E[T_i]$ represents $E[T]$ when $n = i$, and $E[T_\tau]$ means $n \to \infty$. When $n \to \infty$ is the mode with infinite offloading restart, as derived in [43]:

$$E[T_\tau] = \frac{M(\tau) + \tau}{F_o(\tau)} - \tau. \tag{15}$$

When $n = 1$,

$$E[T_1] = M(\tau) + (1 - F_o(\tau))(\tau + E[T_l]). \tag{16}$$

The criterion to decide whether to restart or not can be formulated as $E[T] < E[T_o]$. With (14), this condition can be written as the following inequality:

$$E[T_l] < \frac{\int_\tau^\infty t f_o(t)\mathrm{d}t}{(1 - F_o(\tau))^n} - \sum_{k=1}^{n-1} \frac{M(\tau) + \tau}{(1 - F_o(\tau))^k} - \tau \tag{17}$$

In [46], the restart criterion for $n = 1$ has been derived as

$$E[T_l] < \frac{\int_\tau^\infty t f_o(t)\mathrm{d}t}{1 - F_o(\tau)} - \tau \tag{18}$$

For $n \to \infty$, the restart criterion is

$$\frac{M(\tau) + \tau}{F_o(\tau)} - \tau < E[T_o]. \tag{19}$$

Theorem 1 *When the criterion of exclusive local restart ($n = 1$, Eq. 18) is satisfied, the criterion of the adaptive restart scheme ($n > 1$, Eq. 17 or $n \to \infty$, Eq. 19) is also satisfied.*

Proof As the prerequisite of using offloading is $E[T_o] < E[T_l]$, Eq. (18) can be extended as:

$$E[T_o] < E[T_l] < \frac{E[T_o] - M(\tau)}{1 - F_o(\tau)} - \tau \tag{20}$$

$$\Longrightarrow$$

$$E[T_o](1 - F_o(\tau)) < E[T_o] - M(\tau) - \tau(1 - F_o(\tau))$$

$$\Longrightarrow$$

$$M(\tau) + \tau(1 - F_o(\tau)) < E[T_o]F_o(\tau)$$

$$\Longrightarrow$$

$$\frac{M(\tau) + \tau}{F_o(\tau)} - \tau < E[T_o].$$

So when $n \to \infty$, Theorem 1 is true. Let

$$g_1(\tau) = \frac{\int_\tau^\infty t f_o(t)\mathrm{d}t}{1 - F_o(\tau)} - \tau. \tag{21}$$

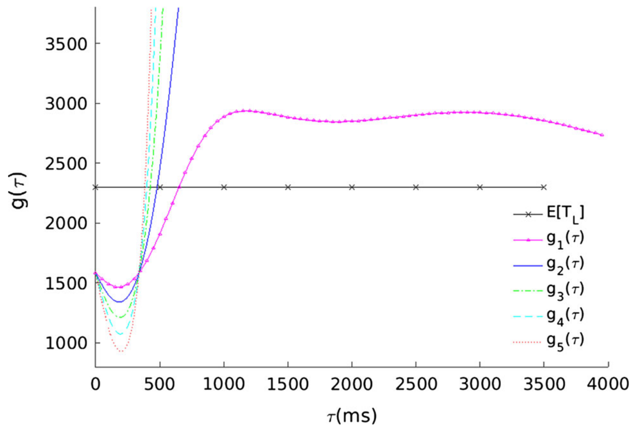


Fig. 9 Restart timeout with 0–4 offloading retries and 1 local retry

When $n \geqslant 2$,

$$g_n(\tau) = \frac{\int_\tau^\infty t f_o(t)\,\mathrm{d}t}{(1 - F_o(\tau))^n} - \sum_{k=1}^{n-1} \frac{M(\tau) + \tau}{(1 - F_o(\tau))^k} - \tau. \tag{22}$$

$$\begin{aligned} g_n(\tau) - g_{n-1}(\tau) &= \frac{\int_\tau^\infty t f_o(t)\,\mathrm{d}t}{(1 - F_o(\tau))^{n-1}} \left(\frac{1}{1 - F_o(\tau)} - 1\right) \\ &\quad - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} \\ &= \frac{E[T_o] - M(\tau)}{(1 - F_o(\tau))^n} F_o(\tau) \\ &\quad - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} \end{aligned} \tag{23}$$

Recalling the result of Eq. (19) and substituting for $E[T_o]$, we have

$$\begin{aligned} g_n(\tau) - g_{n-1}(\tau) &> \frac{\frac{M(\tau) + \tau}{F_o(\tau)} - \tau - M(\tau)}{(1 - F_o(\tau))^n} F_o(\tau) \\ &\quad - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} = 0. \end{aligned} \tag{24}$$

Hence, we have proved that $g_n(\tau) > g_{n-1}(\tau)$, when $n \geqslant 2$. Thus if $g_1(\tau) > E[T_l]$, the inequality of (17) is always true for any $n > 1$, including $n \to \infty$.

Figure 9 shows the result of (22), calculated according to $f_o(t)$ of the bad subset from Table 2. Theorem 1 is also demonstrated in Fig. 9. An interesting point worth mentioning is that when $(M(\tau) + \tau)/F_o(\tau) - \tau = E[T_o]$, τ is at the critical value and $g_n(\tau) = g_{n-1}(\tau) = \cdots = g_1(\tau) = M(\tau)$. When τ is larger than this critical value, $g_n(\tau)$ becomes an increasing function with n at the same τ. Thus if $g_1(\tau) > E[T_l]$, then $g_n(\tau) > E[T_l]$. This proves Theorem 1.

Theorem 2 *When the criterion of restart is satisfied as the inequality (19) holds, the expected task completion time*

$E[T_n]$ decreases with the number of restart n at the same τ.

Proof Using the prerequisite of using offloading $E[T_o] < E[T_l]$, the inequality of (19) can be extended as:

$$\frac{M(\tau)+\tau}{F_o(\tau)} - \tau < E[T_l]. \tag{25}$$

When $n > 1$, we have $E[T_n] < E[T_{n-1}]$:

$$\begin{aligned} &E[T_n] - E[T_{n-1}] \\ &= (1-F_o(\tau))^{n-1}(\tau + M(\tau)) + (1-F_o(\tau))^{n-1}(\tau \\ &\quad + E[T_l])(1-F_o(\tau)-1) \\ &= (1-F_o(\tau))^{n-1}((\tau+M(\tau)) - (\tau+E[T_l])F_o(\tau)) \\ &< (1-F_o(\tau))^{n-1}\left(\frac{(\tau+M(\tau))}{F_o(\tau)} - \tau - E[T_l]\right) = 0. \end{aligned} \tag{26}$$

When $n \to \infty$:

$$\begin{aligned} &E[T_\tau] - E[T_n] \\ &= \frac{M(\tau)+\tau}{F_o(\tau)} - \tau - M(\tau) - \sum_{k=1}^{n-1}(1-F_o(\tau))^k(M(\tau)+\tau) \\ &\quad - (1-F_o(\tau))^n(\tau + E[T_l]), \end{aligned} \tag{27}$$

$\because$

$$\sum_{k=1}^{n-1}(1-F_o(\tau))^k = \frac{(1-F_o(\tau)) - (1-F_o(\tau))^n}{F_o(\tau)}, \tag{28}$$

$\therefore$

$$\begin{aligned} &E[T_\tau] - E[T_n] \\ &= \frac{M(\tau)+\tau}{F_o(\tau)} - \tau - M(\tau) - \frac{1-F_o(\tau)}{F_o(\tau)}(M(\tau)+\tau) \\ &\quad - \frac{(1-F_o(\tau))^n}{F_o(\tau)}(\tau + M(\tau)) \\ &\quad - (1-F_o(\tau))^n(\tau + E[T_l]) \\ &= (1-F_o(\tau))^n\left(\frac{M(\tau)+\tau}{F_o(\tau)} - \tau - E[T_l]\right) < 0. \end{aligned} \tag{29}$$

Thus if inequality (25) holds, we have $E[T_\tau] < E[T_n] < \cdots < E[T_1] < E[T_o] < E[T_l]$. This proves Theorem 2.

5.2 Optimal restart timeout

The optimal restart timeout is the value of τ where $E[T_i]$ is minimal. For comparing the system performance under the adaptive restart scheme with different number of restart, Fig. 10 shows $E[T_i](i = 1–5)$, $E[T_\tau]$ and $E[T_o]$ for the same data

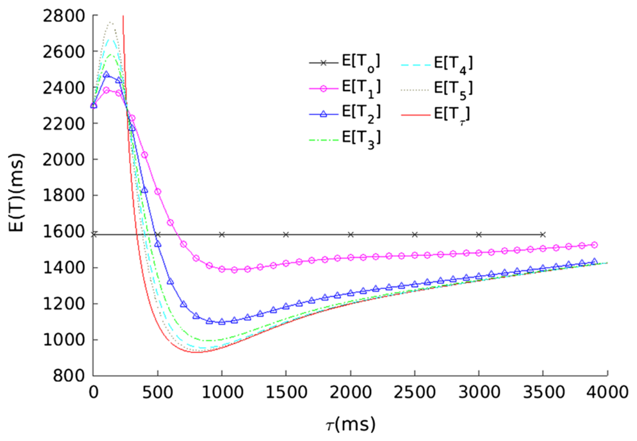


Fig. 10 Expectation of the task completion time versus τ under different restart polices

Table 4 Performance comparison under identical restart timeout

	$E[T_1]$	$E[T_2]$	$E[T_3]$	$E[T_4]$	$E[T_\tau]$
Min	2630	2305	2230	2208	2198
τ	6901	4402	3576	3185	2818
d	0.49	0.87	0.96	0.98	1

$f_o(t)$ of the bad subset from Table 2. As expected $E[T_\tau]$ has the best performance, the minimum of $E[T_\tau]$ is the lowest. The most important observation shown in Fig. 10 is that when $i = 2$, $E[T_2]$ is quite close to the ideal performance of $E[T_\tau]$. To evaluate the performance of different number of restarts, we define a metric to measure the distance between the optimal performance $E[T_\tau]$ and $E[T_i]$. As we have proved that $E[T_i] < E[T_o]$ for $i = 1, 2, \ldots, \infty$, the value of $E[T_i]$ can only vary in the space of $(E[T_\tau], E[T_o])$. So we define $d_i = (E[T_o] - E[T_i])/(E[T_o] - E[T_\tau])$ to measure how close is $E[T_i]$ to $E[T_\tau]$.

From Table 4, we can easily find that the performance of using two restarts can closely approach the best performance of infinite offloading restarts. Because $E[T_2]$ has reached 87 % performance of $E[T_\tau]$, whereas if restarting exclusively with the local execution, its performance $E[T_1]$ reaches merely 50 % of $E[T_\tau]$. Accordingly, we can conclude that the adaptive restart scheme is better than the exclusive local restart. This is because it uses the offloading restart to speed up the task completion. Actually, by applying the local restart, the adaptive restart scheme can also avoid the unpredictable waiting time of redundant offloading restart. We will use experiments to prove this conclusion in Sect. 7.

5.3 Individual restart timeout

In the previous subsection, we have evaluated the performance of the adaptive restart scheme with identical optimal timeout value. In this subsection, applying a different τ_l as the timeout for the local restart is analysed. T' denotes the task completion time when τ_l is allowed.

(one offloading and one local) can reach almost 90 % of the best performance.

$$F(t) = \begin{cases} F_o(t) & 0 \leqslant t < \tau \\ 1-(1-F_o(\tau))(1-F_o(t-\tau)) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ 1-(1-F_o(\tau))^{n-1}(1-F_o(t-(n-1)\tau)) & (n-1)\tau \leqslant t < (n-1)\tau+\tau_l \\ 1-(1-F_o(\tau))^{n-1}(1-F_o(\tau_l)(1-F_l(t-(n-1)\tau)-\tau_l) & (n-1)\tau+\tau_l \leqslant t \end{cases} \tag{30}$$

$$f(t) = \begin{cases} f_o(t) & 0 \leqslant t < \tau \\ (1-F_o(\tau))f_o(t-\tau) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ (1-F_o(\tau))^{n-1}f_o(t-(n-1)\tau) & (n-1)\tau \leqslant t < (n-1)\tau+\tau_l \\ \quad (1-F_o(\tau))^{n-1}(1-F_o(\tau_l) & (n-1)\tau+\tau_l \leqslant t \\ \quad f_l(t-(n-1)\tau)-\tau_l) & \end{cases} \tag{31}$$

As the function (6) and (7), $n \geqslant 2$ in the above function (30) and (30). When $n = 1$, the situation is identical with (8) and (9), so we do not write them out again.

Using the similar expression as Eq. (11), we calculate the expectation of T' as

$$E[T'] = \sum_{k=0}^{n-1}(1-F_o(\tau))^k(M(\tau)+k\tau F_o(\tau)) + (1-F_o(\tau))^{n-1}(1-F_o(\tau_l))((n-1)\tau + \tau_l + E[T_l]) \tag{32}$$

Unfortunately, it is impossible to give a simplified expression for $E[T']$ as for (14). Figure 11 shows $E[T_2']$ for various values of τ with different τ_l. The bottom of the graph indicates the best timeout fraction (parameters in Table 5). Comparing the two Tables 4 and 5, we can find that using different timeout values for the offloading and local restarts individually can improve the performance. The minimum of $E[T_2']$ is less than that of $E[T_2]$. But considering the complexity of the computation for the optimal τ and τ_l, this performance increase is expensive. In addition, in real applications, it is difficult for a mobile device to estimate the optimal τ and τ_l online as it requires to simultaneously sort a two-dimensional matrix. A large amount of time and energy is required for calculating the individual timeout values, but the benefit is limited. Thus, we cannot recommend the use of individual timeouts for the adaptive restart scheme.

Before moving on to the next section, we briefly review the conclusions obtained in this section.

1 Theoretically, infinite offloading restart is the optimal option with the lowest mean completion time. But applying the adaptive restart scheme with two restarts (one offloading and one local) can reach almost 90 % of the best performance.

2 Although triggering the offloading and local restart with individual timeout value can slightly reduce the expected completion time, due to its high complexity for calculating the optimal individual timeout values, we do not implement this scheme in the experiments.

6 Dynamic restart scheme

The procedure of fitting a theoretical distribution and computing the optimal restart timeout from this distribution

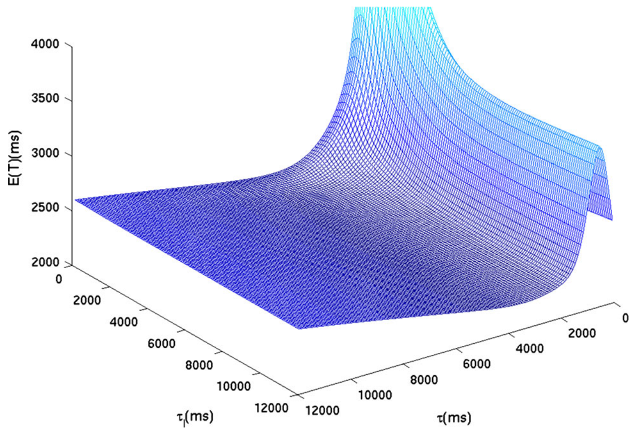


Fig. 11 Expectation of the task completion time versus τ and τ_l

Table 5 Performance comparison with optimal timeout τ and τ_l

	Min	τ	τ_l	d
$E[T_2']$	2284	3767	6901	0.90
$E[T_3']$	2220	3266	6901	0.97

is computationally very expensive. Various algorithms and tools exist for fitting PH distributions to empirical data [17,39,40,45], and the fitted distributions approximate the data in many cases very well. For efficiency reasons we use a direct method [33] to estimate $g(\tau)$ 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.

6.1 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 [25,30,35].

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 initialize 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 offload-

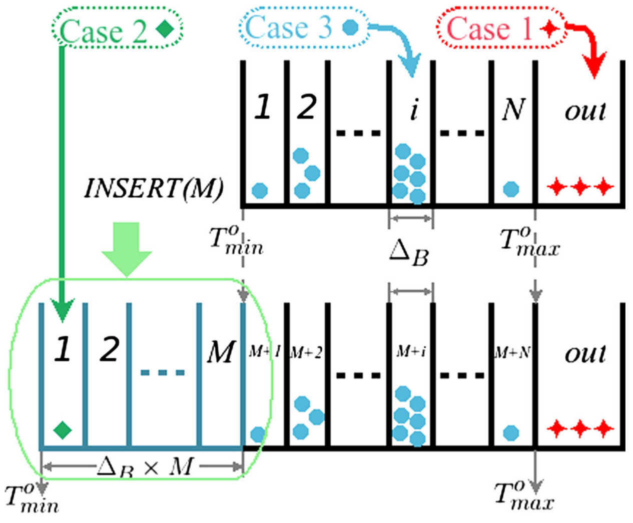


Fig. 12 Recording a new offloading sample

ing 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. Lines 21–24 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. Figure 12 is the illustrative diagram of the three cases.

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.

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
```

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. The mechanism is related to hysteresis as used in the control of queueing systems.

6.2 Asymptotically unbiased ratio estimator

The estimate for the optimal restart timeout is based on the asymptotically unbiased ratio estimator [7]. Using the dynamic histogram proposed in Sect. 5.2, an estimator for $g_n(\tau)$ in Eq. (22) is:

$$\hat{g}_n(\tau_i) = \frac{\sum_{j=i}^{N} N_B[j] \cdot B_{average}[j] + N_{out} \cdot B_{out}}{\left(\sum_{k=i}^{N} N_B[k] + N_{out}\right)\left(1 - \hat{F_o}'(\tau_i)\right)^n} - \sum_{l=1}^{n-1} \frac{\hat{M}'(\tau_i) + \tau_i}{\left(1 - \hat{F_o}'(\tau_i)\right)^l} - \tau_i - T^o_{min}\left(1 - \frac{1}{\hat{F_o}'(\tau_i)}\right)\left(1 - \frac{1}{\left(1 - \hat{F_o}'(\tau_i)\right)^{n-1}}\right) \tag{33}$$

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

$$\hat{F_o}'(\tau_i) = \frac{\sum_{j=1}^{i} N_B[j]}{\sum_{k=1}^{N} N_B[k] + N_{out}} \tag{34}$$

The partial moment $\hat{M}'(\tau_i)$ is estimated as:

$$\hat{M}'(\tau_i) = \frac{\sum_{j=1}^{i} N_B[j] \cdot B_{average}[j]}{\sum_{k=1}^{i} N_B[k]} \tag{35}$$

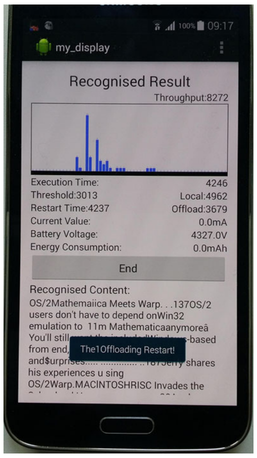
If the maximum estimate $\hat{g}(\tau_i)_{max} > T_l$, the restart condition (18) is fulfilled. Then, an estimate of $E[T]$ provides the optimal timeout.

$$\hat{E}[T]_{\tau_i} = \hat{M}'(\tau_i) + \sum_{k=1}^{n-1}(1 - \hat{F_o}'(\tau))^k(\hat{M}'(\tau_i) + \tau_i) + (1 - \hat{F_o}'(\tau_i))(\tau_i + T_l) + \frac{1 - (1 - \hat{F_o}'(\tau_i))^n}{\hat{F_o}'(\tau)} T^o_{min} \tag{36}$$

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:

$$\hat{M}'(\delta_i) = \frac{\sum_{j=1}^{i} N_B[j] \cdot B_{average}[j]}{\sum_{k=1}^{i} N_B[k]} \tag{37}$$

The optimal restart time can be identified by selecting the value of δ_i, which minimizes $\hat{E}[T]_{\delta_i}$, and the optimal



my_display

Recognised Result

Throughput:8272

Execution Time: 4246
Threshold:3013 Local:4962
Restart Time:4237 Offload:3679
Current Value: 0.0mA
Battery Voltage: 4327.0V
Energy Consumption: 0.0mAh

End

Recognised Content:
OS/2Mathemaiica Meets Warp. . .137OS/2
users don't have to depend onWin32
emulation to 11m Mathematicaanymoreâ
You'll still -based
from end,
and$urpri shares
his experiences u sing
OS/2Warp.MACINTOSHRISC Invades the

The1Offloading Restart!

Fig. 13 User interface of the offloading engine implemented with the dynamic adaptive restart scheme

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.

7 Comparison in practical application

In order to observe and analyse the performance of the adaptive restart scheme in a real application we design an experiment. Using the experiment we show a comparison of five restart modes under different thresholds using the same scenario. Figure 13 shows the user interface of the offloading engine which is implemented with the dynamic adaptive restart scheme.

7.1 Experiment environment

In order to imitate a scenario where the network connection is unreliable, we walk around the main building in our campus carrying the mobile device. The blue line in Fig. 14 shows the route of the device. As shown by the scale on the upper right corner in Fig. 14, the area of this teaching building is about $300 \times 200\,\mathrm{m}^2$ and seamlessly covered by wireless network. When the mobile device is moving, it has to hand-off frequently between different Wi-fi access points (APs). We state that the time interval of switching from one AP to the next is decided by some factors, e.g. the moving velocity of the device, the idle capacity of the next AP and the number of available APs near the device. But in our experiment, we assume that this interval time is a random variable. During the interval, the wireless network is unavailable for the mobile device. Thus, the wireless network connection between the mobile device and the server is unreliable when the device is moving.

The same OCR offloading task introduced in Sect. 3 has been repeated successively while the mobile device was moving. The results were stored in a text file in the mobile device. The memory of the mobile phone used for caching is cleared after each task completion and reused again in the next new task.

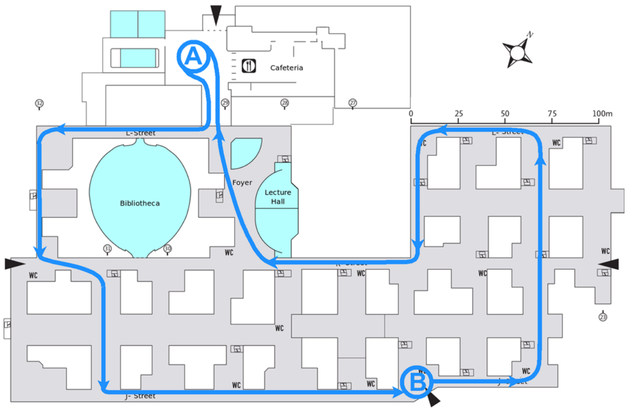


Fig. 14 Plan of the teaching building

7.2 Experimental results

In our experiment, we started from point A as shown in Fig. 14. For initializing the dynamic histogram, we stayed there for 5 min. Then, we walked 10 min to the point B. A second chronograph is used to measure the time in the experiment. Although we cannot accurately reached the point B on time, we guaranteed that the deviation is no more than 15 s. Then we had a rest at the point B for 5 min. During the break, the mobile device connected with an identical Wi-fi AP which covers the area around point B. Even when we move in the $5 \times 5\,\mathrm{m}^2$ area around B, the mobile device did not hand-off to another AP. Thus, in the remaining time, the mobile device kept a reliable network connection to the server. The reason we stayed was to observe the impact of restart on a good network connection in the offloading system.

After the break, we spent 10 min again walking along another route back to point A as shown by the arrow in Fig. 14. The total time of one walk was 30 min. We compared five different restart modes:

A: No restart.
B: Infinite offloading restart, $n \to \infty$.
C: Exclusively local restart, $n = 1$.
D: One offloading restart + local restart, $n = 2$.
E: Two offloading restart + local restart, $n = 3$.

We walked along the same route for each scheme six times and added up the results. The throughput of the five schemes over periods of 5 min is shown in Fig. 15. We defined the throughput as the number of tasks completed in each period. For each scheme, the number of tasks completed by the orig-

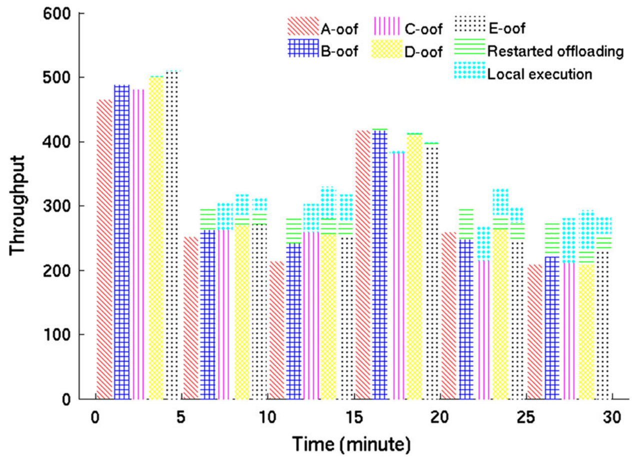


Fig. 15 Throughput of different times in a day

inal offloading (oof), the restarted offloading and the local execution are marked individually in Fig. 15. Surprisingly, the experiment result shows that infinite offloading restart is not the optimum as its throughput is less than the other three restart schemes *C*, *D* and *E*. The explanation for this phenomenon is that when the mobile device is changing Wi-fi AP, sometimes the hand-off process requires tens of seconds. In particular, if the next access point has already connected to a large number of users, the new coming mobile device is hardly assigned sufficient resources to build a reliable connection. Connecting to a heavily loaded AP means that the hand-off time may extend to several minutes. During this time, repeatedly restarting to offload cannot speed up the task completion. Under a slow hand-off, local restart can at least guarantee task completion.

Figure 15 also demonstrates that the throughput of scheme *C* is lower than that of scheme *D*, *E*. This phenomenon follows Theorem 2 in Sect. 5. However, the throughput of scheme *D* is higher than that of scheme *E* because in practical applications successive offloading tries are in many cases not independent. Generally, the failure of the first offloading restart indicates a high probability of the failure of successive offloading restarts. Our experiments give a strong indication of the correlation between successive restarts, at least when network quality is poor. Due to its complexity we did not consider the correlation in our theoretical analysis. In conclusion, the adaptive restart with one offloading retry and one local restart is in this scenario the optimal scheme to increase system performance.

8 Conclusion

In this paper we have extended our previous local restart scheme introduced in [46]. We have proposed a new adaptive restart scheme to improve the performance of mobile offloading systems. Before a task terminates by local restart in the mobile device, the adaptive scheme restarts it first using several offloading attempts. When the number of offloading restarts tries exceeds a given threshold, local restart is launched. Restarting the task again at the appropriate moment can reduce its overall completion time.

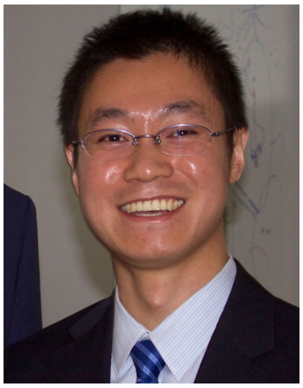
First, we introduced an experiment to illustrate the impact of network delays on mobile offloading. Then, we mathematically derived the optimal threshold and timeout in order to reduce the task completion time. We proposed a dynamic scheme to implement the adaptive restart for the mobile offloading system. In this scheme, a dynamic histogram is used to track the variation of the network quality, and the restart condition and the optimal time are estimated using the histogram.

By theoretically comparing the performance of applying different numbers of offloading retries, infinite offloading restart is proved to perform best, given that successive tries are independent and follow the same probability distribution. However, in practice a limited number of offloading restarts are preferred. Therefore, we have replaced infinite tries with a final local execution. We experimentally explore how many offloading restarts should preceed the final local execution as to optimize the system throughput (and hence the job completion time). We find that after one failed offload attempt local execution leads to higher throughput than another offload try. We assume that this effect indicates correlation of successive tries since most network problems have longer duration. Besides the correlation of successive delays, accelerating the adaptation of restart configuration by quickly updating threshold and timeout also improves the system performance. And the frequency of tuning the histogram has an impact on the update process. In the future we will include correlation, the speed of adaptation and the histogram update frequency in our theoretical study as to better predict the expected benefit of repeated offloading tries.

References

1. Asmussen, S., Fiorini, P., Lipsky, L., Rolski, T., Sheahan, R.: Asymptotic behavior of total times for jobs that must start over if a failure occurs. Math. Oper. Res. **33**(4), 932–944 (2008)
2. Balan, R.K., Satyanarayanan, M., Park, S.Y., Okoshi, T.: Tactics-based remote execution for mobile computing. In: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, pp. 273–286. ACM (2003)
3. Bobbio, A.: Trivedi, K.S: Computation of the distribution of the completion time when the work requirement is a ph random variable. Stochastic Models **6**(1), 133–150 (1990)
4. Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., Bahl, R.: Maui: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile systems, Applications, and Services, pp. 49–62. ACM (2010)
5. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In:

Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation, vol. 2, pp. 273–286. USENIX Association (2005)
6. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer Systems, pp. 301–314 (2011)
7. Cochran, W.G.: Sampling Techniques. Wiley, London (2007)
8. Crovella, M.E., Taqqu, M.S., Bestavros, A.: Heavy-tailed probability distributions in the World Wide Web. In: Adler, R.J., Feldman, R.E., Taqqu, M.S. (eds.) A Practical Guide to Heavy Tails, vol. 1, pp. 3–26. Birkhäuser, Chapman and Hall, New York (1998)
9. Di, S., Robert, Y., Vivien, F., Kondo, D., Wang, C.-L., Cappello, F.: Optimization of cloud task processing with checkpoint–restart mechanism. In: 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12. IEEE (2013)
10. Danilkina, A., Reinecke, P., Wolter, K.: Sfera: a simulation framework for the performance evaluation of restart algorithms in service-oriented systems. Electron. Notes Theor. Comput. Sci. **291**, 3–14 (2013)
11. Deboosere, L., Simoens, P., et al.: Grid design for mobile thin client computing. Future Gener. Comput. Syst. **27**(6), 681–693 (2011)
12. Egwutuoha, I.P., Levy, D., Selic, B., Chen, S.: A survey of fault tolerance mechanisms and checkpoint–restart implementations for high performance computing systems. J. Supercomput. **65**(3), 1302–1326 (2013)
13. Fernando, N., Loke, S.W., Rahayu, W.: Mobile cloud computing: a survey. Future Gener. Comput. Syst. **29**(1), 84–106 (2013)
14. Flinn, J., Park, S.Y., Satyanarayanan, M.: Balancing performance, energy, and quality in pervasive computing. In: Proceedings. 22nd International Conference on Distributed Computing Systems, 2002, pp. 217–226. IEEE (2002)
15. Fourneau, J.-M., Wolter, K., Reinecke, P., Krauß, T., Danilkina, A.: Multiple class g-networks with restart. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, pp. 39–50. ACM (2013)
16. Hargrove, P.H., Duell, J.C.: Berkeley Lab checkpoint–restart for linux clusters. In: J. Phys. Conf. Ser. 46:494 (2006)
17. Horváth, A., Telek, M.: Phfit: a general phase-type fitting tool. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) Computer Performance Evaluation: Modelling Techniques and Tools, pp. 82–91. Springer, Berlin (2002)
18. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
19. Kephart, J.O.: Research challenges of autonomic computing. In: Proceedings of the 27th International Conference on Software Engineering, pp. 15–22. ACM (2005)
20. Kulkarni, V.G., Nicola, V.F., Trivedi, K.S.: On modelling the performance and reliability of multimode computer systems. J. Syst. Softw. **6**(1), 175–182 (1986)
21. Kemp, R., Palmer, N., Kielmann, T., Bal, H.: Cuckoo: a computation offloading framework for smartphones. In: Griss, M., Yang, G. (eds.) Mobile Computing, Applications, and Services, pp. 59–79. Springer, Berlin (2012)
22. Kristensen, M.D.: Scavenger: transparent development of efficient cyber foraging applications. In: 2010 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 217–226. IEEE (2010)
23. Marinelli, E.E.: Hyrax: cloud computing on mobile devices using MapReduce. Technical report, DTIC document (2009)
24. Maurer, S.M., Huberman, B.A.: Restart strategies and internet congestion. J. Econ. Dyn. Control **25**(3), 641–654 (2001)
25. Matias, Y., Vitter, J.S., Wang, M.: Dynamic maintenance of wavelet-based histograms. In: Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00, pp. 101–110, San Francisco, CA. Morgan Kaufmann (2000)
26. Nicolae, B., Cappello, F.: BlobCR: virtual disk based checkpoint–restart for HPC applications on IaaS clouds. J. Parallel Distrib. Comput. **73**(5), 698–711 (2013)
27. Ni, X., Meneses, E., Jain, N., Kalé, L.V.: ACR: automatic checkpoint-restart for soft and hard error protection. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 7. ACM (2013)
28. Nicola, V.F., Trivedi, K.S.: The completion time of a job on multimode systems. Adv. Appl. Probab. **19**(4), 932–954 (1987)
29. O'Donoghue, B., Candes, E.: Adaptive Restart for Accelerated Gradient Schemes. Found. Comput. Math. 1–18 (2013)
30. Poosala, V., Haas, P.J., Ioannidis, Y.E., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. ACM SIGMOD Rec. **25**(2), 294–305 (1996)
31. Ruan, Y., Horvitz, E., Kautz, H.: Restart policies with dependence among runs: a dynamic programming approach. In: Principles and Practice of Constraint Programming-CP 2002, pp. 573–586. Springer (2002)
32. Reinecke, P., Krauß, T., Wolter, K.: Cluster-based fitting of phase-type distributions to empirical data. Comput. Math. Appl. **64**(12), 3840–3851 (2012)
33. Reinecke, P., van Moorsel, A., Wolter, K.: A measurement study of the interplay between application level restart and transport protocol. In: Proceedings of the First International Conference on Service Availability, pp. 86–100. Springer (2004)
34. Satyanarayanan, M.: Pervasive computing: vision and challenges. IEEE Pers. Commun. **8**(4), 10–17 (2001)
35. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: new aggregation techniques for sensor networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 239–249. ACM (2004)
36. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. Pervasive Comput. IEEE **8**(4), 14–23 (2009)
37. Sheahan, R., Lipsky, L., Fiorini, P.M., Asmussen, S.: On the completion time distribution for tasks that must restart from the beginning if a failure occurs. ACM SIGMETRICS Perform. Eval. Rev. **34**(3), 24–26 (2006)
38. Smith, R.: An overview of the tesseract ocr engine. In: ICDAR, vol. 7, pp. 629–633 (2007)
39. Thummler, A., Buchholz, P., Telek, M.: A novel approach for phase-type fitting with the EM algorithm. IEEE Trans. Dependable Secure Comput. **3**(3), 245–258 (2006)
40. Telek, M., Heindl, A.: Matching moments for acyclic discrete and continuous phase-type distributions of second order. Int. J. Simul. Syst. Sci. Technol. **3**(3–4), 47–57 (2002)
41. Van Moorsel, A.P.A., Wolter, K.: Analysis and algorithms for restart. In: First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings, pp. 195–204. IEEE (2004)
42. van Moorsel, A.P.A., Wolter, K: Meeting deadlines through restart. In: MMB, pp. 155–160 (2004)
43. Van Moorsel, A.P.A., Wolter, K.: Analysis of restart mechanisms in software systems. IEEE Trans. Softw. Eng. **32**(8), 547–558 (2006)
44. Wang, Q., Jorba, M.G., Ripoll, J.M., Wolter, K.: Analysis of local re-execution in mobile offloading system. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 31–40. IEEE (2013)
45. Wang, J., Liu, J., She, C.: Segment-based adaptive hyper-Erlang model for long-tailed network traffic approximation. J. Supercomput. **45**(3), 296–312 (2008)
46. Wang, Q., Wolter, K.: Reducing task completion time in mobile offloading systems through online adaptive local restart. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, pp. 3–13. ACM (2015)



Qiushi Wang received the B.E. degree in 2008 and the M.E. degree in 2011, both from Beijing University of Posts and Telecommunications, Beijing, China. He received the Ph.D. degree in the Institute of Computer Science, Free University Berlin, Germany, in 2015. He received the Best Paper award in the 6th ACM/SPEC International Conference on Performance Engineering 2015. His research interests include Mobile Offloading System, Restart Algorithm and Stochastic Modeling.

Katinka Wolter has been professor for dependable systems at Freie Universität, Berlin, since 2012 after having been a lecturer for dependable architectures at Newcastle University for a short period and having held different research positions at Freie Universität and Humboldt University, both in Berlin. She received her Diploma and her Ph.D. from Technical University Berlin in 1996 and 1999, respectively, and her habilitation degree from Humboldt University, Berlin, in 2008. Her research interests are in model-based evaluation of dependable architectures, distributed systems and networks.