Model-based Performance Analysis of Local Re-execution Scheme in Offloading System

Qiushi Wang, Huaming Wu, Katinka Wolter
Department of Mathematics and Computer Science
Freie Universität Berlin, Takustr.9, Berlin, Germany
Email: qiushi.wang@fu-berlin.de, katinka.wolter@fu-berlin.de

***Abstract*—Offloading is a useful approach to save energy and time for mobile devices by migrating heavy computation to remote powerful servers. However, the unreliable wireless network constrains the implementation of offloading applications. The execution continuity is always interrupted by network failures. To deal with this problem, locally re-executing the pre-determined offloading task in the mobile device is a valid method. Challenges arise due to the best trade-off between costs and benefits of Local Re-execution. In this paper, using a Stochastic Activity Network model, we defined three metrics to investigate the performance of Local Re-execution, which is launched by different timeout values. Through comprehensively comparing the simulation results, we further explored the optimal timeout value for activating Local Re-execution, and reached the conclusion that the optimum is mainly controlled by the delay of network recovery.**

***Index Terms*—Modeling; performance analysis; Offloading; Stochastic Activity Network; timeout;**

I. Introduction

A significant improvement of mobile communications has been seen in recent years. Both the hardware processing rate and the flexibility of operating systems are able to undertake some heavier applications which could previously only be run on desktops or servers. However, even though smart mobile devices have undergone a fast development, they are still unable to compete with their desktop siblings. The constraints are obvious. The limited battery capacity prevents a long run time of some intensively computational applications (like image processings or games). To deal with this problem, Offloading is one of the popular technics. It migrates the heavy computation to remote servers through the wireless network. This can not only save the energy but also accelerate the execution time and thus provides a better user-experience. Connecting mobile terminals and remote servers, the wireless network plays an important role in the offloading system. To some degree, it directly affects the performance of the offloading execution. Usually, the offloading does not require a mass data transmission between mobile clients and remote servers. It only needs an overhead of several Megabits to migrate the executing thread [1] or the application state [2] from mobile terminals to remote servers and then get back the results. This tiny delay of data transmission will not impair the application performance.

As the remote cloud servers of large enterprises are supported by better back-up and protection schemes, comparing with the frequently fluctuated wireless network condition, they can be seen as high-availiability systems. In that way, the wireless network becomes the major cause of bringing the unstability in offloading systems. There are two major failure handling schemes. The first one is a halt in the current execution state and waiting for the network condition to satisfy the offloading requirement, then relaunching the offloading task. The other one is as soon as the wireless connection lost, the mobile device immediately re-executes the pre-determined offloading task locally. However, both of the two schemes only adapt extreme scenarios. The former one performs well when the wireless network can quickly recover to the demanding level. But it may wast a large amount of time for waiting if the network suffers from a long time failure. In this case, the latter should be choosen. Although it costs more time and battery energy, the continuity of application execution is maintained. As well, Local Re-execution is not cost-effective in the first scenario. In order to provide a balanced performance in both the scenarios, combining the two schemes with a timeout scheme is a reasonable method.

In this paper, we used SAN(Stochastic Activity Network) [3] models to simulate the execution of offloading systems and computed three metrics: Unstability, Energy Consumption and Throughput to evaluate the performance of the offloading system. The paper is organized as follows: after presenting a survey on related concepts for offloadind and SAN in Section 2, we make a description of the proposed models in section 3. We derive formulas how to compute the three metrics and the normalization method to synchronise them in section 4. Simulations and results are shown in Section 5 while Section 6 concludes the paper.

II. Background and Related Work

As soon as the concept of cloud computing was proposed, it has attracted attentions to integrate mobile devices within the cloud. Partitioning and Offloading are two fundamental steps to implemente the mobile cloud computing. Partitioning tries to reasonably separate the mobile application into several components. Some of them can be executed in remote servers and the others can only be run locally in mobile devices. It has been researched deeply [4]. Offloading has to decide, based on different wireless network conditions, which parts of the movable components should be migrated to execute in remote servers. According to the architectures, different

978-1-4799-0181-4/13/$31.00 ©2013 IEEE

methods (CloneCloud [1], MAUI [2], VM-Based Cloudlets [5]) are used to complete the two jobs.

For analyzing the performance of the offloading in mobile clouds, one of the popular ways is using stochastic models [6]. But these models can only analyze the execution time, and merely consider the influence of wireless network factors like MTTF. In this paper, using SAN, many more system factors, like the processing rate of mobile devices, the delay of data transmission between clients and servers and the workload of offloading tasks were considered. Besides the execution time, by refering some experimental parameters, we derived the formulas to calculate the energy consumption.

SAN was an extension to Petri-net, which has been extensively used for performance modeling to analyze computer and communication systems. In order to represent the timeliness and parallelism of the system in a stochastic setting, several extensions have been proposed like GSPNs and SRNs [7]. SAN was defined with the purpose of facilitating unified performance/dependability evaluation. Timed and Instantaneous activities, the essential core of SAN, are easily used to represent the execution of the modeled system. Another element, gate in SAN, permits greater flexibility in defining enabling and completion rules. It is convenient to be used as representatives of the selection between waiting or Local Re-execution. In brief, SAN is a general modelling technique that can be easily used to analyse the performance of offloading.

III. Performance Analysis Models

There are two steps for evaluating the performance. The first one is using Stochastic Activity Networks (SAN) as modeling technique to determine the measures. The second one is using these measures to calculate the performance by formulas. The models are described in this section while the formulas are introduced in the next section. Our models of the entire offloading system consists of two parts: a network model and an execution state model. The network model is an independent model (Fig.2). It is used to simulate varied conditions of the wireless network during offloading. The execution state model (Fig.3) is used to simulate the execution of offloading in mobile devices. Each state in the offloading process is represented by the corresponding *Marking* in our SAN models, and the running of offloading is represented by *Activities* which control the changes between those states. We also discussed the interactions between the two models and the factors which disturb the completion of offloading tasks.

A. System Description

Based on the proposal in [2], we describe the offloading process as shown in the flow chat Fig.1. It has five states. The function of each state is listed as below:

- Migrating: The mobile device transmits the necessary information to the remote server for offloading tasks. The time spent in this state is related to the network condition and the data size. During this time, data transmission could be interrupted by the network failure. The mobile device has to restart this work after the network recovery.

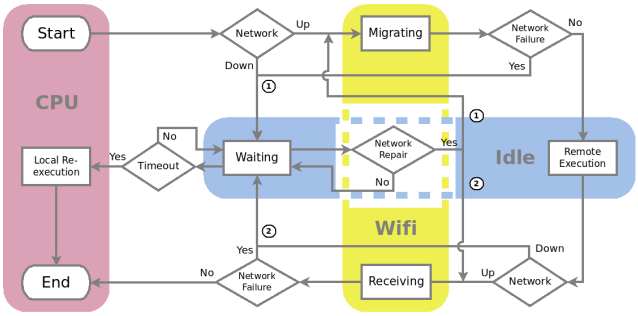

Fig. 1. Flow Chart of the offloading process

- Remote Execution: After the mobile device completing the data transmission, offloaded tasks are executed in the remote server. The mobile device holds on in the current state, and waits for receiving the result from the remote server.
- Receiving: This state is the same as Migrating. When the network condition satisfies the offloading requirement, the mobile device receives the result of completed offloaded tasks from the remote server. If not, the mobile device goes into Waiting state.
- Waiting: When the wireless network fails, the offloading process moves to this state and the mobile device begins to count the waiting time. After the network condition recovered, the data transmission resumes again. But once the waiting time exceeds the timeout limit, the mobile device stops waiting and launches Local Re-execution.
- Local Re-execution: When the mobile device has waited quite a long time for the network recovery. In order to avoid wasting any more time, the pre-determined offloading task is locally executed by the mobile device instead of the remote server.

Both before and during the data transmission (Migrating and Receiving), the condition of the wireless network is monitored. When the network cannot support the data transmission, the mobile device moves to the state of Waiting. There are two inputs for Waiting (① and ②), they come from Migrating and Receiving individually. After the network recovery, the offloading process goes back to the state according to the input source. If the network is not stable, as the connection loses frequently, the mobile device has to wait a long time for a sufficient up-time of the network to complete data transmission. In order to avoid the long waiting time, the task, which has been decided to offload, is re-executed locally in the mobile device. It may need a longer execution time than offloading, but the execution continuity is maintained. Generally, the offloading tasks are mostly intensive. Local Re-execution may quickly use up the capacity of the battery in the mobile device. If the failed network can be repaired in a short time, it is worth waiting for a moment rather than re-executing immediately.

As we known, a mobile device consists of several components like CPU, Storage, Antenna, Battery and so on. During serving the function of each state in the offloading process, the

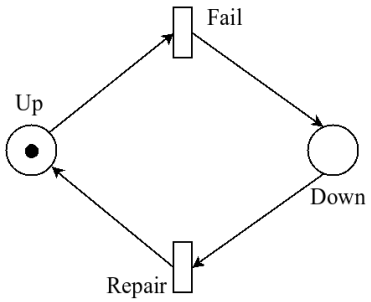

Fig. 2. Wireless Network Model

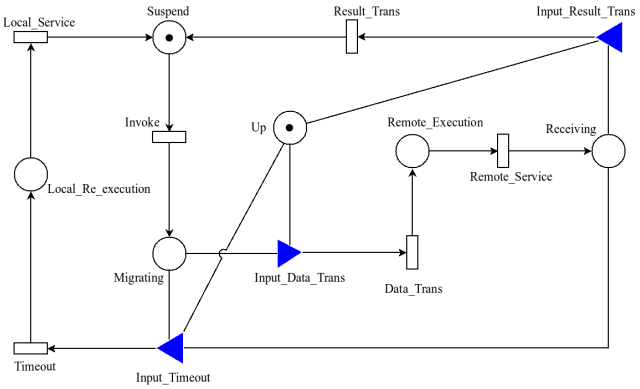

Fig. 3. Execution State Model

utilizations of these components are different. In Migrating and Receiving, the utilizations of Antenna and other components related with wireless communication are high, but CPU does not take too many jobs. Whereas in Local Re-execution, the utilization of CPU is close to 100%, but Antenna is almost not used. Thus, when completing the function of a given state, we consider these components make up a particular module. The power of this module equals the sum of every individual component power. As researched in [8], the energy consumed by each module does not change. Therefor, we classify the states based on the modules used by them. As this paper mainly focusing on theoretical analysis, for simplifying the calculation only three kinds of modules (CPU, WiFi and Idel) are used. It is assumed that the energy consumption of the states belong to the same module are equal. Migrating and Receiving are grouped into WiFi as they mainly use wireless components. Waiting and Remote Execution are belong to Idle, because most components are in the idle state when the mobile device is waiting. In Local Re-execution, CPU of the mobile device undertakes many intensive computations, it consumes a lot of energy, thus it is seperated as a single category.

B. Wireless Network Model

As can be seen in Fig.2, this simple model only consists of two places *Up* and *Down*, two activities *Fail* and *Repair*. During transmitting the data for offloading tasks, it is easy to judge the wireless network between two states: up or down. If the network staisfies the requirements, it is in the up state(Token in *Up*), offloading can be executed smoothly. If not, the network is in the failed state(Token in *Down*). Through the two activities *Fail* and *Repair*, the network condition turns from one state to the other after halting for a randomly distributed time, with the probability density function $f(t)$. In this paper, we use the exponential distribution as the $f(t)$ of two activities *Fail* and *Repair*.

C. Execution State Model

Based on the offloading process described before, the places in Execution State Model are corresponding to the states in the process. It is easy to find this relation between Fig.1 and Fig.3. Execution State Model consists of five places, the meanings of their markings are as below:

- *Suspend*: The mobile device is executing locally or in the idle state. When the token returns back to this place, it denotes the end of an offloading task execution. Then after activating *Invoke*, the token moves out again and indicates the beginning of a new offoading task.
- *Migrating*: The marking of this place may denote two states: Migrating and Waiting. If the marking of *Up* is 0, token in *Migrating* represents the mobile device is in the Waiting state. Otherwise, it represents in the Migrating state. Until *Data_Trans* is activated, the data transmission is successful. Before that, the mobile device could moves into the Waiting state again, once the marking of *Up* changes from 1 to 0 as the enabling condition of *Data_Trans* is broken (Table I).
- *Remote_Execution*: Tokens in this place denotes that the mobile device is in the Remote Execution state and waits for the completion of the offloading task.
- *Receiving*: As the same as *Migrating*. It denotes the mobile device could be in both states of Receiving or Waiting. After *Result_Trans* is activated, the result of the offloading task from remote servers is received successfully.
- *Local_Re_execution*: Its marking denotes that the offloading process is in the state of Local Re-execution. After a local service time, modeled by activating *Local_Service*, the pre-determined offloading task is completed locally.

This model also shares the same place *Up* with the previous network model. Through the input gates, it controls the enabling condition of the three activities, *Data_Trans*, *Result_Trans* and *Timeout*, which change the marking of *Migrating* and *Receiving*. If the wireless network is not in the up state, the two activities *Data_Trans* and *Result_Trans* cannot be activited. The token only stays in *Migrating* or *Receiving*. When the wireless network returns to the up state, these activities would be activated again. However, the activation of *Data_Trans (Result_Trans)* only moves the token from the place *Migrating (Receiving)* to the place *Local_Re_execution (Suspend)*. The marking of *Up* is not changed in this model, it only depends on Wireless Network Model.

The activity *Timeout* is used to control the timeout value for triggering Local Re-execution. The waiting time is recored at the beginning of the offloading process going into the Waiting state. Once the network is repaired, the enabling predicates of

TABLE I
ENABLING PREDICATE OF THE INPUT GATES

Gate	Predicate	Function
Input_Data_Trans	#*Up* > 0 & #*Migrating* > 0	#*Migrating* = 0; #*Remote_Execution* = 1;
Input_Result_Trans	#*Up* > 0 & #*Receiving* > 0	#*Receiving* = 0; #*Suspend* = 1;
Input_Timeout	(#*Up* == 0 & #*Migrating* > 0), or (#*Up* == 0 & #*Receiving* > 0),	#*Local_Re_execution* = 1; If(#*Migrating* > 0) then #*Migrating* = 0; else if (#*Receiving* > 0) then #*Receiving* = 0;

refers to the number of tokens in the given place

Data_Trans or *Result_Trans* are satisfied again. The offloading process returns back to the Migrating or Receiving states. If the duration of a network failure exceeds the timeout value, *Timeout* is activated and the token moves into *Local_Re_execution*, which represents the offloading process is in the Local Re-execution state. Normally, the timeout value should be fixed. Definitions of those parameters are shown in Table II.

IV. COMPUTING PERFORMANCE

In this paper, we focus on the performance comparison between different timeout values, which postpones Local Re-execution. Three metrics are choosed to measure the performance: Unstability, Energy Consumption and Throughput. They are not only used independently, but also after normalization combined by calculating the geometric distance from the best value of each metric. The data of Unstability and Throughput are directly got from the results of model simulations. Energy Consumption has to be calculated with the power parameters (Table III).

A. Unstability

The aim of this metric is to estimate the delay caused by the network failure which brings the interruption in the data transmission. Unstability is defined as the probability of an offloading task meets the connection failure during transmitting data with remote servers. In the model it is represented as the token halting in *Migrating* or *Receiving*, when the network is in the down state. As shown in Table I, it is the same as the enabling predicate of *Input_Timeout*.

$$Pr_{unstability} = Pr((\#Migrating = 1 \vee \#Receiving = 1) \wedge \#Up = 0) \quad (1)$$

B. Throughput

Throughput *H*, which reflects the efficiency of the system, is always one of the most important metrics for any computer system. In this paper, we defined *H* as the number of offloading tasks served in a particular model time. In the model, it is represented as the number of *Invoke* activations.

TABLE II
PARAMETERS FOR EACH ACTIVITY (EXPONENTIAL DISTRIBUTION)

Name	Parameter	Meaning
Invoke:	$\lambda_{arrival}$	$T_{arrival} = 1/\lambda_{arrival}$ is the mean time before the next offloading request arrival.
Data/Result_Trans:	λ_{trans}	$T_{trans} = 1/\lambda_{trans}$ is the mean time to complete the data transmission.
Remote_Service:	λ_{server}	$T_{server} = 1/\lambda_{server}$ is the mean service time of the offloaded task in the remote server.
Timeout:	$P \times (T_{server} + 2 \times T_{trans})$ *	*P* is the percentage of the entire execution time of an offloading task, it controls timeout values.
Local_Service:	$\frac{\lambda_{server}}{N}$	*N* is the multiple of processing rate of the remote server with respect to the mobile device.

* In the offloading process, it experiences two data transmission states: *Migrating* and *Receiving*. As the $f(t)$ of *Data_Trans* is the same as *Result_Trans*, the numerator is 2.

C. Energy Consumption

In [8], Aaron Carroll made detailed measurements of the power consumption of each module in a mobile device. Although the power consumptions of the same module in different devices are not equal, generally the ratios between these modules do not change. So, for convenient theory analysis, we calculated Energy Consumption by directly referring to the parameters given in Table III. The value of Energy Consumption in each state equals the holding time, as the token stays in the corresponding place, multiples the power of the module (CPU, WiFi and Idle), which the state belongs to. The energy consumed in Execution State Model is:

$$E'_{re} = T_{trans} \times p_{wifi} + T_{idle} \times p_{idle} + T_{re} \times p_{cpu} \quad (2)$$

T_{trans} is the time spent in Migrating and Receiving. It includes the time wasted in the interrupted data transmission. T_{idle} consists of two parts T_1 and T_2, T_1 is the holding time in Remote Execution. T_2 is the holding time in Waiting. As being restricted by the timeout value, T_2 has an upper limit as $P \times (1/\lambda_{server} + 2/\lambda_{trans})$. Local Re-execution is launched when the waiting time exceeds this limitation. T_{re} is the time used for Local Re-execution, which has the positive correlatioin with the processing rate of the mobile device N and the remote service time of the offloading task T_1. T_s is the total model time.

$$T_{trans} = Pr((\#Migrating = 1 \vee \#Receiving = 1) \wedge \#Up = 1) \times T_s \quad (3)$$

$$T_{idle} = T_1 + T_2 \quad (4)$$

$$T_1 = Pr(\#Remote\_Execution = 1) \times T_s \quad (5)$$

$$T_2 = Pr_{unstability} \times T_s \quad (6)$$

$$T_{re} = N \times T_1 \quad (7)$$

TABLE III
PARAMETERS OF ENERGY CONSUMPTION IN THE MODULES [8]

State	Module	Power
Migrating/Receiving	Wifi	$p_{wifi} = 900mw$
Waiting	/	$p_{idle} = 250mw$
Local_Re_execution	CPU	$p_{cpu} = \begin{cases} 900mw, & N = 10 \\ 450mw, & N = 20 \\ 280mw, & N = 30 \end{cases}$

The power of CPU is linked with their processing rates. The faster the mobile CPU is, the more energy it requires. In this paper, we only took three values of CPU power into consideration (as shown in Table III, N = 10, 20 and 30, the larger N means the slower processing rate of CPU). Actually under different workloads, the power of CPU should have different values. But we only tried to illustrate how to use our models to calculate the energy used in the offloading re-execution. So when the mobile device moves to the state of Local Re-execution, we assumed that the workload of CPU keeps the same.

$$E_{re} = E'_{re}/H \tag{8}$$

Actually, the total Energy Consumption is also related to Throughput H in a particular model time. To exclude this influence, the average Energy Consumption of each offload task E_{re} is calculated.

D. Synthetical Performance Analysis

The order of magnitudes and the units of three metrics are quite different. In order to synthetically analyze them, at first, we used normalization to transform all the data of the same metric into the range of [0,1]. The transform formula is

$$y = \frac{x - Min(X)}{Max(X) - Min(X)} \tag{9}$$

After transforming, we found that under the same timeout value, the reaults of three metrics make up a three-dimensional vector $\langle u_i, e_i, h_i \rangle$ (u_i: Unstability, e_i: Energy Consumption, h_i: Throughput). The performance of each timeout value is defined as the geometric distance of this vector from the best one. To Unstability, it is expected to be the lower the better as the same as Energy Consumption. But to Throughput, if the system can accomplish more tasks in a period, it has a better performance. Therefore, the best vector is $\langle 0, 0, 1 \rangle$, and the distance is calaulated as:

$$A = \sqrt{w_1 \times u_i^2 + w_2 \times e_i^2 + w_3 \times (1 - h_i)^2} \tag{10}$$

(w_1, w_2, w_3) is the weight vector for the three metrics. It can be adapted for different application scenarios. But in this paper, we only analyzed the result of equal weight vector (1,1,1). Through the geometric distance, it provides a direct view of the comparison between the performances of different timeout values.

TABLE IV
MODEL PARAMETER VALUES USED IN THE SIMULATION

parameter	List of values (in seconds)
Arrival rate $\lambda_{arrival}$	1/10
Data Transmission time T_{trans}	2, 5, 8, 10, 16, 20
Remote service time T_{server}	5, 10, 20
Mean times to failure T_{fail}	100, 500, 1000, 2000
Mean times to repair T_{repair}	12 values between 10 and 120
Percentage of timeout P^+	21 values between 0% and 200%
Speed multiple (cloud/mobile) N	10, 20, 30 *

$^+$ refer to Table II item *Timeout*
* these values refer to the experimental results in [5]

V. SIMULATION AND RESULT

In order to explore the effects of the restart timeout under different workloads, hardware capabilities and network conditions, we simulated our models using Mobius [9] by using Lagged Fibonacci random number generator with the Seed of 31415. Performance was investigated through the simulations by varying the values of related parameters, which represents various system configurations and offloading environments. The complete set of simulation parameters is provided in Table IV. We fixed these paramenter values only based on assumptions, which are made as a matter of experience and convenience. All the parameter combinations have been simulated with a model time of 86400s(24 hours).

The performance is calculated by formula (10). Fig.4 depicts the performance comparison of various P with different T_{repair}. Since the geometrical distance is used as the criterion to evaluate the performance, the bottom in the picture is the most desirable. We also drew the performance of no re-execution in the end of the axis P. Apparently it is far from the best vector. Thus in this way, it has also been testified that Local Re-execution is more advantageous to deal with the network failure. Then we tried to find out, under the particular T_{repair}, if there is an optimal P^*. Even it can not provide the best performance in all T_{trans}, it is still able to acquire acceptable results of the three matrics no matter how long

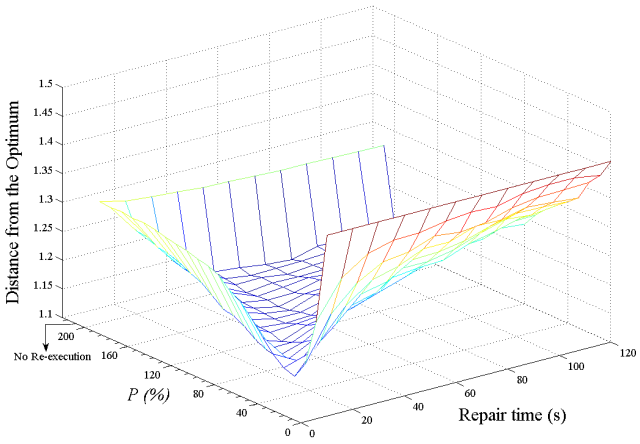

Fig. 4. Performance comparison with T_{repair} and P, T_{fail} = 500s, T_{server} = 10s, T_{trans} = 10s, N = 20.

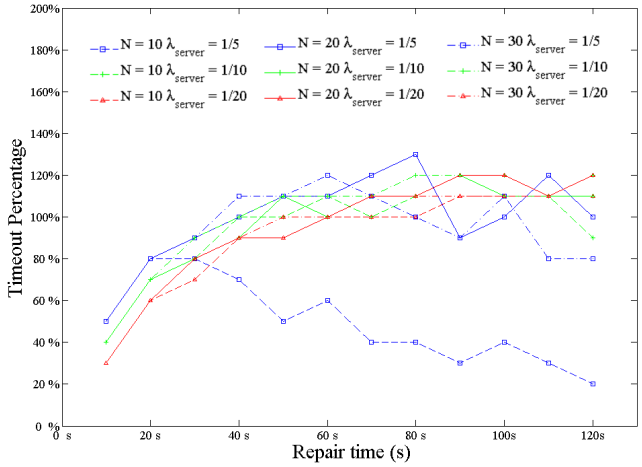

Fig. 5. Optimal P with λ_{repair} under different λ_{server} and N, T_{fail} = 1000s, T_{trans} = 2s∼20s

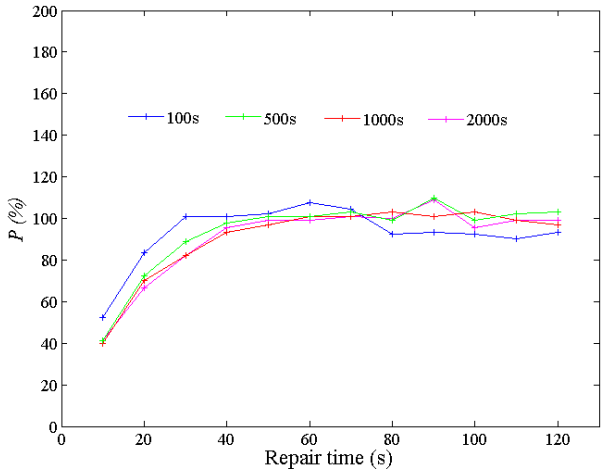

Fig. 6. Optimal P with λ_{repair} under different λ_{fail}.

T_{trans} is. This P^* can be viewed as the default optimum for the offloading process with an unknown workload. For this target, we summed up the performance values of all T_{trans} at the same P. The reaults show that the shape of the amount of performance also resembles a valley like Fig.4.

Fig.5 shows the changing of the optimal P^* with T_{repair} under different mobile device capabilities N and workloads of the offloading task λ_{server}. At a first overview, the optimal P^* has an increasing trend with T_{repair}. But they stay around the value of 100% after T_{repair} passing a specific threshold ($T_{repair} > 50s$), besides an unusual case $N = 10$, $\lambda_{server} = 1/5$. In this case, the mobile device has a powerful capability (low N), it can complete the task in a short time without consuming too more energy for Local Re-execution. Although the little longer time of Local Re-execution decreases Throughput, it is offset by the saved waiting time. Throughput could even be improved when T_{repair} is quite long. Therefore, when the offloading task is not heavy and the mobile device is fast, it is better to launch Local Re-execution as quickly as possible, rather than wasting a long time for the network recovery.

It is easy to find in Fig.5, most of the optimal P^* under different N and λ_{server} distribute around a given curve. We tried to identify this curve by calculating the mean of those optimums in the same T_{repair}. The extreme case ($N = 10$ and $\lambda_{server} = 1/5$) is excluded. Fig.6 shows the results of different T_{fail}. Under different T_{fail}, the curves of the optimal P^* are nearly the same. This demonstrates that T_{fail} has no effect on the optimal P^*. Another point is that, when $T_{reapir} \leq 30s$, the curve has an increasing trend. When $T_{reapir} \geq 40s$, the optimum stays around 100%. Thus, despite there are some minor fluctuations, this value can be considered as the default optimum for offloading tasks.

VI. Conclusion

This paper proposed a SAN model for performance analysis of failure handling schemes in offloading systems. By adapting the most commonly used timeout mechanism, an efficient failure handling scheme combining Local Re-execution and network recovery waiting has been introduced. Our models allow to comprehensively analyze the effects of different parameters on the performance. Through simulation, the analysis results indicate that the optimum is not affected by the failure rate of the wireless network but decided by its repair rate. When the network repairs quickly, the optimal timeout value increases with it. When the network needs a long time to repair, the optimum equals the offloading time. In summary, even these simulation results are theoretical, they are still worthful during investigating the effects of various system and network parameters on the failure handling scheme in offloading.

References

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

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

[3] W. Sanders and J. Meyer, "Stochastic activity networks: Formal definitions and concepts," *Lectures on Formal Methods and PerformanceAnalysis*, pp. 315–343, 2001.

[4] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[6] R. Gabner, H. Schwefel, K. Hummel, and G. Haring, "Optimal model-based policies for component migration of mobile cloud services," in *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 195–202.

[7] J. Muppala, G. Ciardo, and K. Trivedi, "Stochastic reward nets for reliability prediction," *Communications in reliability, maintainability and serviceability*, vol. 1, no. 2, pp. 9–20, 1994.

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

[9] D. Daly, D. Deavours, J. Doyle, P. Webster, and W. Sanders, "Möbius: An extensible tool for performance and dependability modeling," *Computer Performance Evaluation. Modelling Techniques and Tools*, pp. 332–336, 2000.