Performance and Data Traffic Analysis of Mobile Cloud Environments

Pinheiro, Thiago; Airton Silva, Francisco; Fe, Iure; Kosta, Sokol; Maciel, Paulo

Published in:
2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)

DOI (link to publication from Publisher):
10.1109/SMC.2018.00695

Creative Commons License
CC BY 4.0

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Pinheiro, T., Airton Silva, F., Fe, I., Kosta, S., & Maciel, P. (2019). Performance and Data Traffic Analysis of Mobile Cloud Environments. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 4100-4105). Article 8616692 IEEE Signal Processing Society. https://doi.org/10.1109/SMC.2018.00695

Performance and Data Traffic Analysis of Mobile Cloud Environments

Thiago Pinheiro ∗, Francisco Airton Silva †, Iure Fé ∗, Sokol Kosta ± and Paulo Maciel ∗
∗ Informatics Center, Federal University of Pernambuco (UFPE), Recife, Brazil
† Laboratory PASID, Federal University of Piauí (UFPI), Picos, Brazil
± Center for Communication, Media and Information Technologies, Aalborg University, Copenhagen, Denmark
Emails: {tfs3, isf2, prmm}@cin.ufpe.br, faps@ufpi.edu.br, sok@cmi.aau.dk

***Abstract*—Mobile Cloud Computing (MCC) is a technique for increasing the performance of mobile apps and reducing their energy consumption through code and data offloading. Building an MCC infrastructure is a difficult task due to its inherent complexity and the involvement of different components. This paper proposes an approach for estimating applications' performance and data traffic volume generated by tasks offloading. This work proposes a Stochastic Petri Net (SPN)-based formal framework to represent the partitioning of applications in a method-call level. Our framework considers the available network bandwidth to send and receive tasks to the cloud. The modeling strategy represents the use and sharing of the actual available bandwidth for offloading operations. The approach enables designers to plan and tune MCC architectures based on *Mean Time to Execute* (MTTE) and *Throughput* estimation. Using our strategy it is possible to estimate the impact of the bandwidth variation on the application's MTTE and *Throughput*. In addition, the strategies proposed in this work may be adapted to support MCC applications in real time providing on-the-fly probabilistic performance predictions. One case study was performed to evaluate the approach. Our proposed approach has proven to be feasible and it highlights the most appropriate strategies for offloading.**

***Index Terms*—mobile cloud, performance evaluation, data traffic evaluation, stochastic petri nets, ctmc**

I. Introduction

Mobile Cloud Computing (MCC) is a paradigm that increases the performance of mobile apps and reduces their energy consumption through offloading technique [9]. The offloading process sends tasks to be processed on remote servers in the cloud. The first step in performing task offloading is to split an application into different *parts*. Next, there is a decision of which ones are most convenient for remote processing. Tasks may be partially or completely offloaded, depending on the application requirements.

Offloading does not come for free. Cloud service providers charge their clients for resource usage. A wrong offloading decision may lead a company to financial losses. The higher the resource usage, the higher the amount to be paid to the provider. Method-call offloading is a partitioning strategy that enables to split a code into multiple parts. Deciding which method to partition is not an easy task. It is necessary to analyze many possible scenarios.

This paper proposes an approach for estimating the performance and data traffic volume generated by tasks offloading to clouds. For that aim, we defined twofold steps. We propose a Stochastic Petri Net (SPN) [14] modeling strategy to represent the application source code structure. The modeling strategy represents the use and sharing of the available network bandwidth (BW) for offloading operations. Next, the proposed models are used to estimate two performance metrics of the whole application, the *Mean Time to Execute* (MTTE) and *Throughput*. The MTTE corresponds to the execution time of the application as a whole. The throughput corresponds to the number of requests per time unit made by each user. We propose a mobile cloud model to predict data traffic volume applying SPNs.

The remainder of the paper is organized as follows. Section II highlights the main concepts about Method-Call Offloading and SPNs; Section III presents the proposed strategy; Section IV details the case study to support the proposal; and finally, Section V traces conclusions, stressing future directions.

II. Background

This section presents some concepts about Method-Call Offloading and SPNs to support the understanding of the proposed approach.

A. Partial Method-Call Offloading

An alternative to the Full Method-Call Offloading [10], is the Partial Method-Call Offloading [8]. Instead of all methods, only a subset of them is offloaded. Data dependencies must be observed to perform a correct partition. For example, considering the methods *m1()* and *m2()*, two decisions should be made: **Where to execute *m1()* and *m2()*?**

Combining these possibilities (mobile device or the cloud), four scenarios may be exploited (see Table I). The number of possibilities increases proportionally to the application complexity. In a real-world scenario, the number of combinations of method-calls to offloading may be very large. Such variety makes harder the software engineer's work when deciding the most appropriate offloading distribution. Besides, other aspects, such as the mobile device current CPU and energy consumption level, may influence the decision. Assuming that an application with many active users uses a cloud service, a wrong offloading decision may lead to financial losses. Therefore, companies should plan their MCC offloading strategies balancing performance and use of remote resources.

Table I: Scenarios to Method-Calls Executions

Possibility	***m1()***	***m2()***
Scenario #1	mobile	mobile
Scenario #2	mobile	cloud
Scenario #3	cloud	mobile
Scenario #4	cloud	cloud

B. *Stochastic Petri Nets*

Petri Nets (PNs) are a powerful modeling tool that can be used to represent concurrent, asynchronous, distributed, parallel, deterministic, and stochastic processes [14]. PNs define a specification technique that allows a mathematical and graphical representation, and it has analytical mechanisms that enable verification on the properties and correctness of modeled systems. Stochastic Petri Nets (SPNs) are an extension of Petri Nets [12]. SPNs associate a stochastic delay to each timed transition. Thus, PNs become probabilistic, being described by a stochastic process. SPNs may be isomorphic for Continuous Time Markov Chains (CTMC) and, consequently, they can provide performance measures [7].

III. Estimating Performance and Data Traffic

Companies in some situations need to balance system performance and resource consumptions to find the most appropriate strategy that meets the requirements of their projects. As the user base of an offloadable app grows, the higher may be the consumption of some remote resources. Applications' performance and data traffic are key elements in the choice of an appropriate offloading strategy. More precisely, this paper seeks to answer the following questions:

1) How to calculate the throughput and MTTE of a set of method-calls — that may represent a system functionality — using SPNs?
2) How to estimate the impact of the available bandwidth variation on the application's MTTE?
3) How to estimate the data volume that will be transferred during the offloading process of a set of method-calls?

A. *Network Performance and Communication Time*

Network performance is a key factor that has a direct impact on the performance of the entire MCC application. However, it is a difficult task to estimate precisely at design time the network conditions in which applications will be used. Real life network conditions may vary abruptly.

Latency variation over wireless networks is hard to predict. As the wireless channel congestion increases, the latency deteriorates rapidly. Packet loss is another factor that impacts on TCP throughput performance. This work considers the expected TCP throughput between the device and the cloud for tasks offloading. A developer may implement a strategy to estimate the actual bandwidth available during the application execution. In this context, there are many works with the aim of evaluating MCC offloading traffic considering the overall networking aspect and resulting network-induced constraints [5]. For example, based on the actual available bandwidth estimated and the performance prediction performed using the strategy proposed in this work, mobile apps may decide how the processing will be executed.

Thus when planning an MCC application, developers should consider that the expected bandwidth may vary within a specified limit. For example, if the application detects that the available bandwidth is less than the lower-limit set value, then the processing is performed locally. Based on the actual bandwidth resources available, it is possible to use the well-known Equation 1 to estimate the communication time.

$$CT = \frac{datasize}{BW} \tag{1}$$

B. *Execution Time (MTTE)*

MTTE corresponds to the average time to finalize the processing of a set of method-calls. Figures 1 and 2 present an example of SPNs for computing MTTE. MTTE is the expected time to reach an absorbing state. A state of an SPN is absorbing whether it is impossible to leave it (i.e., *P(#FINISH = 1)*). It means a deadlock marking has been reached. MTTE is based on the probability that the processing of a system functionality has been completed. MTTE is the average time for a number of tokens to reach the place *FINISH* given they were in place *START* at time instant zero. SPN models can be evaluated either by numerical methods or by simulation [13].

Figures 1 and 2 demonstrate a simple representation using SPN of one functionality with only one method-call. Let us first look at the SPN representation that corresponds to the local method-call (see Figure 1). The SPN model comprises three places and two transitions. The first transition (*trigger_-time*) is immediate. It means that the transition has zero as its delay value. The second transition (*processing_time*) is a General Time High-level Transition. It represents the time to processing the respective method-call.

The SPN pattern that represents an offloadable method-call has two new places and two new transitions (see Figure 2). Transition *offloading_time* represents the time spent to execute offloading. Transition *receiving_time* represents the time spent to receive the result sent by the cloud. These transitions are depicted by a gray rectangle, and the model is later refined by assigning probabilistic distribution parameter values to respective transitions. If, on the one hand, such transition is refined by poli-exponential distributions [6], the SPN can be evaluated either by numerical analysis or by simulation. On the other hand, simulation should be carried out.

The models evolved by transformation of high-level transitions into exponentially distributed timed transitions. It allows assigning average delays to respective timed transitions. Such transformation of transitions and delays assignments allow the SPN models to be solved. From here, for simplicity, all timed transitions will have exponential enabling times. However, they may adopt other probabilistic distributions as well as deterministic values.

Moment matching [6] could also be applied to obtain poly-exponential distributions [15]. By adopting moment matching, the planner may estimate what exponential-based probability distribution best fits the mean. Additionally, moment matching generates more accurate models, which can still be numerically evaluated. If none poly-exponential distributions are adopted, simulations should also be adopted.

Such SPN patterns may originate other models to evidence data dependency between method-calls of any source code arrangement. The SPN modelling pattern evolved to calculate MTTE of distinct scenarios. Figure 3a demonstrates the SPN that represents two serial method-calls. Figure 3b shows the SPN that represents two parallel method-calls. The pattern embraces general features common in concurrent systems. The place *SYSTEM_INACTIVE* when having one token means that the system is idle. The timed transition *T0* receives the delay to start the processing of method-calls when there is a token in the place *SYSTEM_INACTIVE*. When there is no delay, *T0* becomes an immediate transition. In a real-world context, multiple combinations can be derived taking into account the application's method-calls and the modelling patterns presented in Figures 1 and 2.

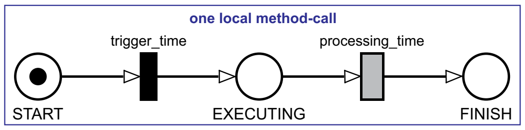


Figure 1: Basic SPN Representation of One Application with Only One Local Method-Call Using Absorbing State

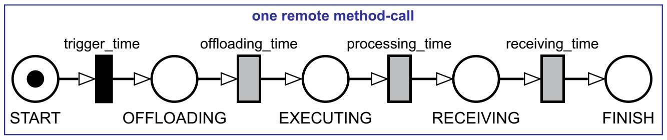
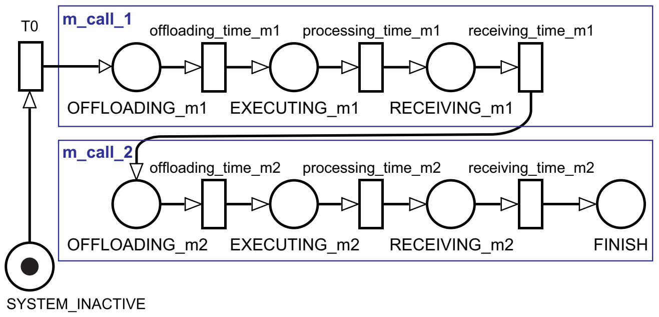


Figure 2: Basic SPN Representation of One Application with Only One Offloadable Method-Call Using Absorbing State

The mean processing time and communication time of each evaluated method are the base for MTTE calculation. In the models presented, the *processing_time_m1* and *processing_time_m2* transitions receive the mean processing times of the methods *m_call_1* and *m_call_2*, respectively. If the application's method under analysis is an offloadable method, it is necessary to obtain the number of bytes transferred to send tasks and to receive the remote results.

In this work, we consider that there is a specific bandwidth allocated to offloading operations, as well as to receive the remote results. More specifically, the developer should consider bandwidth variation for a more accurate estimate (see Section III-A). Equations 2 and 3 consider the probability of there being tokens in the *offloading* place (variable O_{mj}) as well as in the *receiving* place (variable R_{mj}) for other method-calls other than *mi*, respectively. $BW_{offloading}$ represents the actual bandwidth allocated for tasks offloading – in bits/s.

(a) SPN with Absorbing State Used to Calculate MTTE of an Application With Two Serial Method-Calls



(b) SPN with Absorbing State Used to Calculate MTTE of an Application With Two Parallel Method-Calls

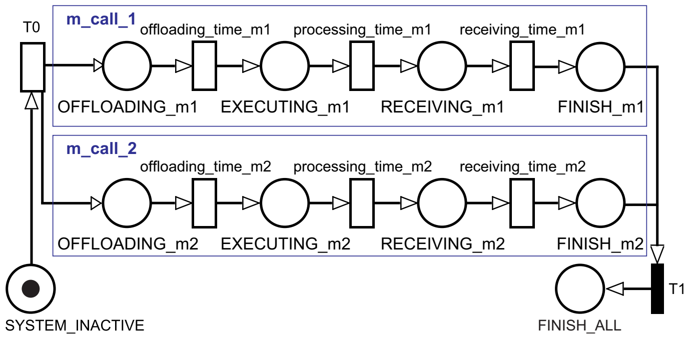


Figure 3: SPNs Representing Two Offloadable Method-Calls

$BW_{receiving}$ represents the actual bandwidth allocated to receive the remote results – in bits/s. Thus, if other methods are using the allocated bandwidth, the bandwidth allocated for the operation (i.e. offloading or receiving) is divided among the methods that are using the network for the same operation.

Equations 4 and 5 calculate the communication time to transfer an amount of data taking into account the actual bandwidth allocated to the evaluated method mi. $datasize\_o_{mi}$ represents the amount of data transferred to offload the method mi – in bits. $datasize\_r_{mi}$ represents the amount of data received as the result of the remote processing of the method-call mi – in bits. Equations 4 and 5 are assigned as the mean delay value of transitions *offloading_time_mi* and *receiving_time_mi* of the method-call *mi*, respectively. In the evaluation process, developers must convert Equations 4 and 5 to the syntax of the stochastic evaluation program used.

After that, a transient analysis on the model obtains the MTTE. A tool such as the Mercury, TimeNet, GreatSPN or SHARPE may execute this analysis. These tools generate the state space of the SPN model and create the corresponding CTMC. Then, the calculations described in the next section are performed to obtain the required metrics.

$$BWO_{mi} = \left(\frac{1}{1 + \sum_{j \neq i}^{n} P\left\{O_{mj} > 0\right\}} \right) \times \left(\frac{BW_{offloading}}{1000} \right) \tag{2}$$

$$BWR_{mi} = \left(\frac{1}{1+\sum_{j\neq i}^{n} P\{R_{mj} > 0\}}\right) \times \left(\frac{BW_{receiving}}{1000}\right) \tag{3}$$

$$offloading\_time_{mi} = \frac{datasize\_o_{mi}}{BWO_{mi}} \tag{4}$$

$$receiving\_time_{mi} = \frac{datasize\_r_{mi}}{BWR_{mi}} \tag{5}$$

1) Mean Time to Absorption (MTTA): Time-dependent metrics are obtained through transient evaluation. The measure of interest is the Mean Time to Absorption (MTTA). To calculate the MTTA, the model must have at least one absorbing state. In this work, the absorbing state is reached when the model is in the *FINISH* state. Figure 4 presents the CTMC that represents the elapsed time to finish the processing for an offloadable method-call.

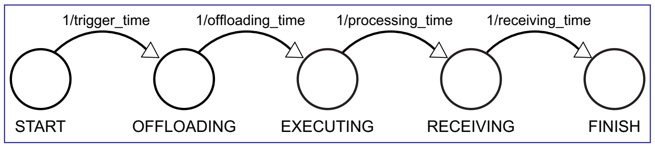


Figure 4: CTMC of One Application with Only One Offloadable Method-Call With Absorbing State

The behavior of the CTMC can be described by the Kolmogorov equation given the initial probability vector $\pi(0)$ (see Equation 6). Equation 7 gives the expected total time the CTMC spends in state i during the interval [0, t). The time spent before absorption can be calculated restricted to the states of the set of non-absorbing states (N) by $lim_{t\to\infty}L_N(t)$. Thus, $L(t)$ satisfies the Equation 8 where $\pi_N(0)$ is the vector $\pi(0)$ restricted to the states in the set N. Q_N is the infinitesimal generator matrix restricted to the non-absorbing states. Finally, MTTA may be described as Equation 9 [4].

$$\frac{d\pi(t)}{dt} = \pi(t)Q \tag{6}$$

$$L_i(t) = \int_0^t \pi_i(x)dx \tag{7}$$

$$L_N(\infty)Q_N = -\pi_N(0) \tag{8}$$

$$MTTA = \sum_{i\in N} L_i(\infty) \tag{9}$$

C. Throughput and Data Transfer Volume

The throughput (Tp) represents the number of executions per unit of time of a set of method-calls. Tp is obtained by computing the expected value of tokens at a place, multiplied by the inverse of the transition delay [11]. Now, as illustrated in Figure 5, the model needs two transitions to allow it to return to the initial state when workload execution is complete. Such SPN pattern may be extended to evidence the method-calls data dependency of any application.

The throughput may be calculated considering two possibilities: Single Server Semantics (SSS) and Infinite Server Semantics (ISS). Equation 10 calculates the throughput according to the SSS strategy, and Equation 11 for ISS. The variable i represents the weight of the arc that connects the place *INACTIVE* and the subsequent transition *T0*. The variable i may vary until N, where N is the highest enabling degree of the subsequent transition at the place marking $m(INACTIVE) = i$.

$$Tp = P(m(INACTIVE) >= i) \times \frac{1}{Time} \tag{10}$$

$$Tp = \left(\sum_{i=1}^{N} P(m(INACTIVE) = i) \times i\right) \times \frac{1}{Time} \tag{11}$$

Using the Equation 12 it is possible to obtain the total transferred bytes (*TTB*). *Time* represents the evaluation period in milliseconds. *Bytes* represents the volume of data transferred in each request.

$$TTB = Tp \times Time \times Bytes \times Users \tag{12}$$

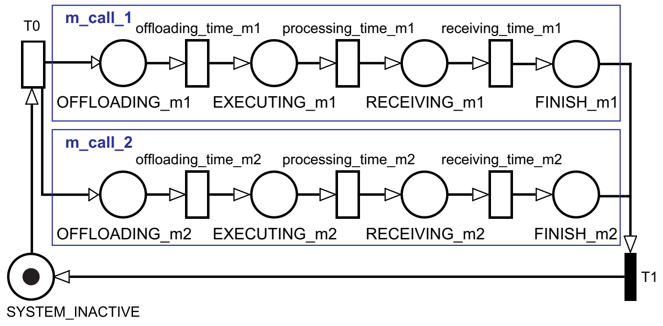


Figure 5: SPN without Absorbing State Used to Calculate Throughput of an Application With Two Parallel Method-Calls

IV. Case Study

This section presents a case study observing the execution time metric using an application for reducing images color.

A. Color Reduction Application

We implement and analyze an image processing mobile application following the principles of method-call computation offloading [8]. The implementation uses a simple client-server architecture with Remote Method Invocation (RMI). The relevant parts of the offloading source code are presented in Listing 1.

Application_A resides on the mobile device and its method-calls are dependency free (lines 5 to 6). That is, the methods may be executed in parallel. If the method-call is offloaded to the cloud, it means that the app makes image processing calls to the server by passing one input (original images). In this case, the app connects to one virtual machine and then calls the method *reduceColor* in the server side. Thereafter, the processed image returns to the mobile device. Table I reveals the method-calls distribution scheme of the application.

Both client and server side adopt the Open Source Computer Vision Library [3] and *JavaCV* [2]. We have implemented the computing vision example of Picture's Colour Reduction [1]. This example transforms images by decreasing the number of colors depending on the picture's size.

```

public class Application_A{
 public List<Image> reduceColor(Image img1, Image img2){
  List<Image> results = new ArrayList<Image>();
  results.add(reduceColor(img1));  /* m_call_1 */
  results.add(reduceColor(img2));  /* m_call_2 */
  return results;
 }
}

public class Server{
 public Image reduceColor(Image image) {
  // JavaCV code supressed
 }
}
```

Listing 1: Client and Server Classes—Image Processing.

1) The Evaluation: Table I demonstrates the evaluated scenarios. The testbed executed all method-calls local and remotely (using one VM as offloading target). Through a controlled experiment, the collected input parameters for each method-call were (see Tables II and III): *(i)* the processing time to process the image and *(ii)* the number of bytes sent and received. As we can see, processing times are lower when the code is running in the cloud. The experiment executed and monitored 50 times each scenario. At the end of the process, the testbed collected the mean values of them.

Table II: Registered Processing Times per Method-Call

Method-Call	Device	Cloud
m1()	140,969 ms	26,644 ms
m2()	260,296 ms	42,612 ms

Table III: Transferred Bytes per Method-Call

Method-Call	Sent Bytes	Received Bytes
m_call_1 (Image 1)	4.05 MB	1.35 MB
m_call_2 (Image 2)	8.43 MB	1.76 MB

The analysis evaluated both the application's MTTE as well as the data traffic generated during a period of 30 days for 1,000 active users. Each user makes requests at a rate of $1/(21,000,000\ ms)$ exponentially distributed. We have considered that the available BW may vary within a specified limit (see Table IV) impacting the MTTE in each variation.

First, it is necessary to analyze the MTTE of each scenario. Figure 3b represents the SPN model of the application. Transitions *processing_time_m1* and *processing_time_m2* are exponential. They store the mean processing times of the method-calls *m1()* and *m2()*, respectively. We have evaluated the four scenarios through the combining of the four processing time values. Equations 4 and 5 were used as the mean delay value of the transitions that represent the offloading and result receiving processes, respectively. We have calculated MTTE taking into account the worst-case BW requirement tolerated by the application as well as the expected network scenario (see Table IV). MTTE will be within these limits. Among other factors, MTTE will vary because of the inherent latency of wireless networks. Table V presents the respective MTTEs.

Table IV: Bandwidth Variation (in Megabits/s)

	BW Variation	
Operation	**Minimum Bandwidth**	**Expected Bandwidth**
Offloading	0.5	2.0
Receiving	1.0	3.0

Table V: MTTE of the Experiment

	Result (MTTE)	
Scenario	**Minimum Bandwidth**	**Expected Bandwidth**
#1	309,821 ms	309,821 ms
#2	241,997 ms	165,587 ms
#3	283,783 ms	265,706 ms
#4	255,970 ms	101,249 ms

The next step is to identify the expected throughput for each user in each scenario using the SPN model presented in Figure 5. Transition *T0* receives the request rate which is equal to $1/(21,000,000\ ms)$. A stationary analysis obtains the throughput using the Equation 10 (see Table VII).

Now, let us estimate the total transferred bytes (*TTB*) generated by each scenario using the Equation 12. Table VI presents the number of bytes transferred for processing one user request in each scenario. Table VIII demonstrates the total bytes transferred in the evaluated period.

We can extract some conclusions by analyzing the results. The performance of scenarios #2, #3, and #4, when executed on the minimum BW requirements, was close to the performance achieved when processing the entire application locally

Table VI: Transferred Bytes for Each Scenario

Scenario	Sent Bytes	Received Bytes	Total Bytes
#1	-	-	-
#2	8.43 MB	1.76 MB	10.20 MB
#3	4.05 MB	1.35 MB	5.40 MB
#4	12.48 MB	3.11 MB	15.60 MB

Table VII: Throughput for Each User in Each Scenario

	Throughput	
Scenario	**Executions/ms**	**Executions/month**
#1	4.6926×10^{-8}	121
#2	4.7246×10^{-8}	122
#3	4.7025×10^{-8}	121
#4	4.7390×10^{-8}	122

Table VIII: Total Transferred Bytes for Each Scenario

		Total Transferred Bytes / month		
		Sent Bytes	**Received Bytes**	**Total Bytes**
Scenarios	**#1**	-	-	-
	#2	1,009.06 GB	211.16 GB	1,220.21 GB
	#3	482.18 GB	160.88 GB	643.06 GB
	#4	1,498.06 GB	373.93 GB	1,871.99 GB

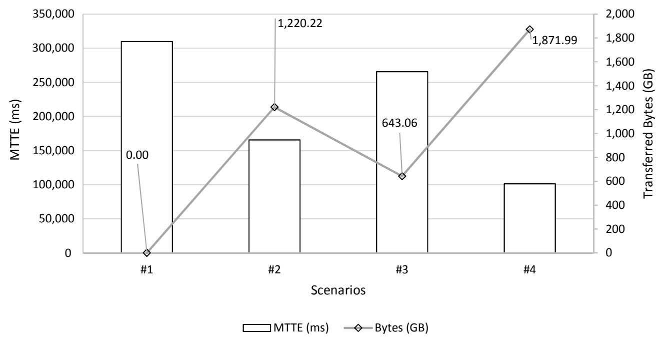


Figure 6: Comparing MTTEs and Volume of Transferred Data

(see Table V). In scenarios #2 and #4 the actual BW available for offloading and receiving had a significant impact on the MTTE of the whole application. The higher the BW, the lower the MTTE. The method *m2()* is the heaviest, so it must be processed in the cloud. If *m2()* is processed locally, maybe it is not advantageous to only perform the offloading of method *m1()* - #3. The performance gain is small. Scenario #4 is most suitable but the volume of data transferred in the period is high (see Figure 6). Thus justifying the adoption of scenario #2 as offloading strategy.

Figure 7 shows the impact of the offloading BW variations on the MTTE. For sake of conciseness, we did not consider variation in the BW to receive the remote results (3.0 Mb/s). The higher the actual BW, the shorter the communication time. The offloading BW variation had little effect on the MTTE of the scenario #3. A large portion of the whole execution time was spent on local processing. On the other hand, scenario #4 was the most sensitive in relation to BW variation. All method-calls were processed remotely. Scenario #4 transferred more data than the other ones. The larger the amount of data being transferred, the higher the impact of the BW variation.

V. Conclusion

In this paper, we proposed an Stochastic Petri Net (SPN)-based formal framework to represent the partitioning of MCC applications in a method-call level. Through our modeling strategy, it is possible to evaluate performance and requests making by mobile users. Our framework considers the available bandwidth to send and receive tasks to the cloud. In this way, representing the communication time to transfer data between the mobile device and the cloud. Besides, we proposed an approach that supports data traffic evaluation using SPNs. Our approach highlights the most suitable options considering specific scenarios. Going forward, we plan to

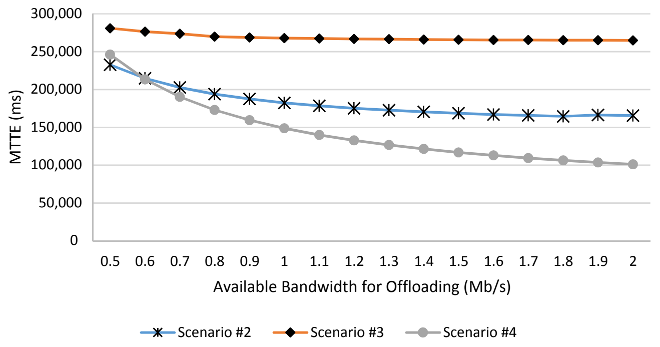


Figure 7: Bandwidth for Offloading and MTTEs

perform more complex experiments and evaluate more metrics, such as energy consumption, and apply the ideas presented herein to other contexts, such as the use of wearable devices in MCC for real-time performance predictions.

References

[1] Colour reduction. http://tinyurl.com/pwq8j44, 2015. Accessed: 2015-07-28.

[2] Javacv. https://github.com/bytedeco/javacv, 2015. Accessed: 2015-07-28.

[3] Opencv. http://opencv.org/, 2015. Accessed: 2015-07-28.

[4] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA, 2006.

[5] N. Cordeschi, D. Amendola, and E. Baccarelli. Reliable adaptive resource management for cognitive cloud vehicular networks. *IEEE Transactions on Vehicular Technology*, 64(6):2528–2537, June 2015.

[6] A. Desrochers, R. Al-Jaar, and I. C. S. Society. *Applications of petri nets in manufacturing systems: modeling, control, and performance analysis*. IEEE Press, 1995.

[7] R. German. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[8] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE INFOCOM*, pages 945–953, March 2012.

[9] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, Apr. 2010.

[10] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '01, pages 238–246, New York, NY, USA, 2001. ACM.

[11] P. Maciel, K. S. Trivedi, R. Matias, and D. S. Kim. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, chapter Dependability Modeling. Premier Reference Source. Igi Global, 2011.

[12] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989.

[13] R. Nelson. *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer Science & Business Media, 2013.

[14] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[15] B. Silva, P. R. M. Maciel, A. Zimmermann, and J. Brilhante. Survivability evaluation of disaster tolerant cloud computing systems. In *Proc. Probabilistic Safety Assessment & Management Conference*, 2014.