Efficient Mobile Computation Offloading with Hard Task Deadlines and Concurrent Local Execution

Peyvand Teymoori, Terence D. Todd and Dongmei Zhao
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, CANADA
Email: {teymoorp,todd,dzhao}@mcmaster.ca

George Karakostas
Department of Computing and Software
McMaster University
Hamilton, Ontario, CANADA
Email: karakos@mcmaster.ca

***Abstract*—This paper considers the problem of algorithmic efficiency in mobile computation offloading with Concurrent Local Execution (CLE). Online energy optimal algorithms can be developed when CLE is used to guarantee hard task deadlines while offloading over Markovian wireless channels. Unfortunately, these algorithms often have a high computational complexity, which prohibits their use in online mobile implementations. Three algorithms are introduced to reduce this complexity: Markovian Compression (MC), Time Compression (TC) and Preemption Using Continuous Offloading (Preemption-CO). MC and TC reduce the state space of the offloading Markovian process, by using a novel notion of geometric similarity, or by running an optimal online offloading algorithm in periodic time steps. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every time-slot is based on non-preemptive calculations. Our simulations show that, by applying these methods, the running times of the algorithms can be significantly reduced without suffering unreasonable performance degradation compared with the optimal energy performance.**

***Index Terms*—Cloud computing, mobile computation offloading, energy efficiency, time efficiency, hard task deadline constraints, concurrent local execution.**

I. Introduction

Mobile computation offloading (MCO) is a mechanism where devices can choose to execute tasks remotely, rather than running them locally on the device itself [1]. This is especially useful for devices with limited energy resources, since the energy needed for task execution can be incurred by the remote cloud server. The decision to offload a task however, is not always straightforward, since the offloading itself will incur both wireless communication energy and latency costs. This is particularly true when the mobile device must interact with the cloud over stochastic transmission channels and/or network conditions. In [2], this issue was studied and an energy model was proposed that considers both mobile computation and communication energy components using statistical inputs. This work assumed that the wireless channel remains unchanged throughout the entire computation offload. In more general scenarios, the wireless channel may evolve randomly during the computation offload.

Offloading over stochastic channels is particularly difficult when the execution completion time of the task to be offloaded must satisfy hard time deadlines. Reference [3] flagged execution time constraints as an important criterion for many interactive applications, and has highlighted the problem of achieving this under stochastic channel conditions. In [4], energy optimal mobile cloud computing was considered assuming random wireless channels but without hard task execution deadlines. In [5], mobile CPU frequency scheduling and transmit power control was used to mitigate wireless channel effects and control task offload execution times. Since this approach cannot always ensure that hard task execution deadlines are met, a parameter was introduced that sets the rate at which deadlines can be violated.

Reference [6] studied *1-Part offloading*, i.e., once the offload starts, it occurs contiguously in one piece without interruption. It introduced *concurrent local execution* (CLE) as a method of enforcing hard task execution time constraints, and presented a provably optimal on-line mobile computation offloading algorithm. CLE guarantees that deadlines are satisfied by allowing simultaneous local task execution during the offload, if it may be needed to complete the task execution on time. This mechanism ensures that deadlines are met even in worst-case situations, such as where the wireless channel suffers a complete outage during the offload. CLE is the method adopted in the current paper.

Mobile device energy can sometimes be further improved by splitting the task upload into multiple parts [6]. This allows the mobile device to adapt to channel conditions at the end of each upload part, provided that the added latency is acceptable. *Preemptive Offloading* is a generalization of this, where the decisions to continue the upload are made at the start of each time-slot packet transmission. Although provably optimal offloading algorithms exist for general Markovian wireless channels and for multi-part offloading, their high computational complexity prohibits their application in practice. In this paper we introduce ways of mitigating this complexity using three different mechanisms: Markovian Compression (MC), Time Compression (TC) and Preemption Using Continuous Offloading (Preemption-CO). The new methods are based on a time-dilated absorbing Markov chain (TDAMC) that was used to define the optimal offloading decisions in [6]. The TDAMC size dictates the time complexity of on-line offloading algorithms. In MC, the TDAMC is reduced by

aggregating its states using a novel method based on the notion of geometric similarity. In TC, the TDAMC is traversed in aggregated time steps, so that algorithm computation is reduced. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every time-slot is based on calculations performed on the (much smaller) 1-Part TDAMC. By applying these methods, the running times of the algorithms can be significantly reduced without suffering unreasonable performance degradation. A wide variety of comparisons are made that illustrate their performance.

II. System Model

A mobile device generates computational tasks that can either be executed *locally*, or can be offloaded over a stochastic communications channel to a cloud server for *remote* execution. Following a common convention (cf. [6] and the references therein), the stochastic communication channel is modelled as a Markov Chain, which transitions amongst states of different bit-rates in every time-slot, and is known to the mobile device.

For every task, it is assumed that its hard completion deadline t_D, number of bits to be uploaded S, remote execution time T_{exec}, and result downloading time T_{down} are known at its release time-slot. Due to the stochastic channel, the time to upload the data from the mobile device to the cloud server T_{up} is a random variable, dependent on the evolution of the channel state as a given upload occurs. For simplicity, we will assume that T_{exec} and T_{down} are both deterministic, and communicated to the mobile device by the cloud server. We study two scenarios for remote execution: The task is either uploaded as a single contiguous block (*1-Part offloading*), or preemption is allowed, i.e., at any time-slot we can decide whether uploading (some of) the task happens or not (*Preemptive offloading*). In order to guarantee the task completion before t_D, a *concurrent local execution (CLE)* model [6] is used, i.e., the task starts local execution at time-slot $t_L = t_D - T_L + 1$, where the local execution time T_L is known, as is the local execution energy consumption E_L. If remote execution completes before local execution finishes, the latter is abandoned.

III. Problem Formulation, Optimal & Approximate Solutions

We are interested in developing on-line algorithms which solve the *CLE offloading problem*, i.e., given the CLE setting of Section II, decide whether to upload part of the task or not at every time-slot, so that the expected task energy consumption is minimized. Note that the decision to upload is made once (if at all) for 1-Part offloading, while it is made repeatedly (if at all) for Preemptive offloading.

In order to develop optimal algorithms for the CLE offloading problem, [6] incorporated both offloading and time in a new Markov Chain, called *a time-dilated absorbing Markov chain ($TDAMC$)*, to model the task uploading evolution, when it starts at the current time-slot t and is done contiguously (for 1-Part offloading), or preemptively (for Preemptive offloading). Recall that we assume prior knowledge of the channel Markovian states, and the transition probabilities P_{ij} for all states i, j. An example of a $TDAMC$ for a 2-state (i.e., Gilbert-Elliot) channel and 1-Part offloading can be seen at the top of Figure 1: The channel goes through two states G, B (i.e., Good or Bad channel conditions), with bit-rates B_{max}, B_{min}, respectively, with transition probabilities $P_{GG}, P_{GB}, P_{BG}, P_{BB}$. $TDAMC(t)$ is constructed by (i) unrolling the evolution of the stochastic channel Markov Chain from the current time-slot t, up to absorbing states indicating the time of task execution completion, and (ii) uploading at every state according to its bit-rate (in case of 1-Part offloading), or branching according to whether the decision for uploading at the current time-slot is made or not. In Figure 1. the $TDAMC(1)$ root state $G_{1,1}$ indicates that the current $t = 1$ channel state is G, and the task is being uploaded with bit-rate B_{max}. Then ($t = 2$) the channel either transitions to B with probability P_{GB}, or remains in G with probability P_{GG}, and the $TDAMC(1)$ transitions to $B_{2,1}$ or $G_{2,1}$, respectively. The evolution of the $TDAMC(1)$ continues in the same fashion, with its states layered as $G_{t,l}, B_{t,m}$ for time-slot t, and $l, m = 1, 2, \ldots$ In general, $TDAMC(t)$ is a layered tree-like Markov Chain, starting with the current state as the root at time-slot t, and going through layers corresponding to time-slots $t+1, t+2, \ldots$ up to time-slot t_D+1, or earlier absorbing states if offloading was finished earlier than t_D.

For the Preemption offloading case, the initial ($t = 1$) $TDAMC(1)$ is enhanced to be a Markov decision process as follows: For every time $1 \leq t \leq t_D + 1$, we define a set of states (X_t, S_t), where X_t is a channel state and $0 \leq S_t \leq S$ is the number of task bits that are uploaded up to t. Let B_{X_t} be the bit rate of channel state X_t. The $TDAMC(1)$ states are again arranged in layers for $t = 1, 2, \ldots, t_D + 1$. The set of actions contains two actions, a_0, a_1, corresponding to not uploading, or uploading, respectively. A state (X_t, S_t) branches to a_0 and a_1; then a_0 branches to states (X_{t+1}, S_t) with probabilities $P_{X_t, X_{t+1}}$, and a_1 branches to states $(X_{t+1}, S_t + \min\{B_{X_t}, S - S_t\})$ with the same probabilities. States of the form (X_t, S) branch only to action a_0 (no uploading). At layer $t = 1$ there is only one state $(X_1, 0)$, where X_1 is the initial channel state, while the states at layer $t_D + 1$ are absorbing. $TDAMC(t)$ is similarly constructed for any current time-slot t.

Using classical Markovian stopping theory, [6] proved that the following simple on-line algorithm solves the CLE offloading problem *optimally* for 1-Part offloading (and the same can be shown for Preemptive offloading[1]): At every time-slot t, compare the expected energy cost of starting uploading at t, to the expected energy cost if we wait to check again at $t+1$, by using Dynamic Programming (DP) on $TDAMC(t)$ and $TDAMC(t+1)$ respectively; if the former is less than the latter, then upload at t.

[1]The Dynamic Programming for Preemptive offloading is more complicated due to the fact that $TDAMC$ states also record the remaining bits to be uploaded, but exactly the same arguments go through.

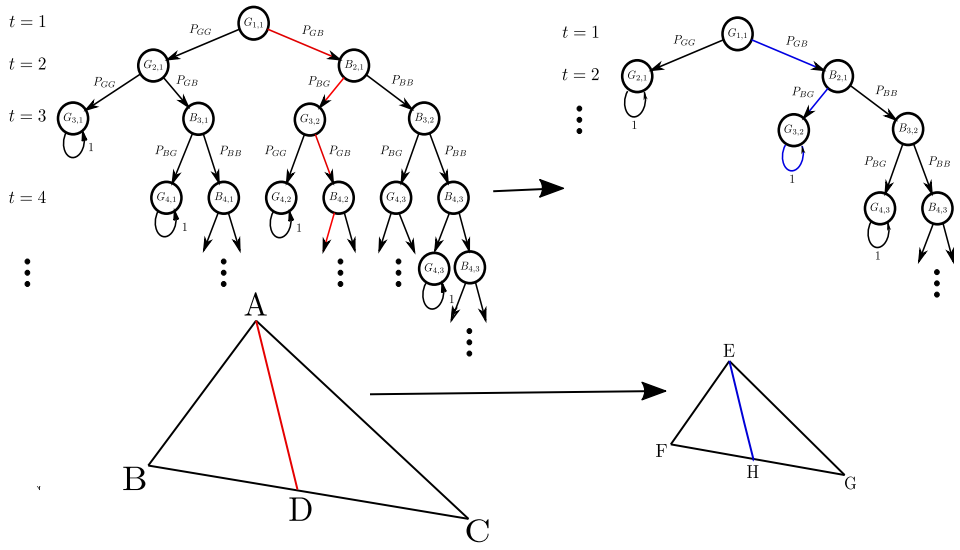


Figure 1: Markovian Compression of the original TDAMC (left) to a smaller TDAMC^{approx} (right).

The DP running time depends on the size of $TDAMC(t)$, since the DP recursion subproblems correspond to $TDAMC(t)$ states. As a result, the optimal algorithm becomes impractical even for simple channel models, such as the 2-state Gilbert-Elliot model; this phenomenon is even more pronounced in the Preemptive offloading case, since the $TDAMC(t)$ states are much more numerous, because they have to also record the remaining task bits to be uploaded.

More specifically, we identify three sources of inefficiency when implementing the optimal offloading algorithm:

A. The size of the $TDAMC$ as a Markov Chain (i.e., number of states and transitions).
B. Running the optimal algorithm at *every* time-slot, imposing a large computational load on the mobile device.
C. In Preemptive offloading, several different uploading time-slots must be picked (instead of a single uploading time-slot for 1-Part offloading). Hence, every state of the $TDAMC$ has to record not only the channel state at a particular time, but the remaining task bits to be uploaded, in order to accurately capture the multiple uploading time-slot combinations.

We address the forbiddingly high running time of the optimal algorithm by designing algorithms addressing each one of these factors. We will evaluate their performance, separately or combined, in Section IV.

A. Markovian Compression (MC)

The large size of the $TDAMC$ in the optimal algorithm is the main reason for the high DP calculation of expected energy consumption. In effect, the DP traverses recursively all possible root-to-leaf tree-paths, and collects the energy spent on each path, weighted by the path probability, in order to compute the total mean energy consumption (the exact recursive process for 1-Part offloading is described in detail in Section VI of [6], and it is similar for Preemptive offloading). If one is to replace the original $TDAMC$ with a smaller $TDAMC^{approx}$, then the latter must approximate well this energy computation, i.e., its (much fewer) paths must be of about the same expected mixture of bit-rates, as in the original $TDAMC$ paths. The key observation on how to do this, is motivated by a geometric analogy (see Figure 1).

We create a new channel Markov Chain model (which will generate $TDAMC^{approx}$) with the same number of states as the original Markov Chain that generated $TDAMC$. In order to determine the new state bit-rates and transition probabilities in $TDAMC^{approx}$, we sort the original $TDAMC$ paths from the shortest to the longest (see Figure 1, left). Recall that each path corresponds to the upoading of S bits along its states, so the left-most (i.e., shortest) path consists of only highest bit-rate states, and the right-most (i.e., longest) path consists of only lowest bit-rate states. Intuitively we would like $TDAMC$ and $TDAMC^{approx}$, seen as 'triangles', to be similar in the following sense: The energy consumption on shortest paths AB, EF should equal the ratio of energy consumption on longest paths AC, EG, while the corresponding number of states ratios should also be equal. Hence, in order for triangles (ABC) and (EFG) to be similar with a scaling factor of l_{MC}, we scale up all the original state bit rates by a factor of l_{MC}; these are the new bit-rates for the channel Markov Chain generating $TDAMC^{approx}$. Next, focusing again on paths AB, EF or AC, EG, we observe that the transitions in the new Markov Chain are in fact transitions in the original Markov Chain, transitioning from the last state of a group of l_{MC} states with bit-rates B_{min} or B_{max}, respectively, to the first state of the next group. Hence, we set the transition probabilities of the new Markov Chain to be *equal* to the original transition probabilities, so that a path AD's probability ends up away from AB's probability by about the same amount that its similar path EH's probability ends up away from EF's probability. To summarize, if P, B are the original transition matrix and state bit-rate vector, then $P^{approx} = P, B^{approx} = l_{MC} \cdot B$ for the new approximate channel Markov Chain. After $TDAMC^{approx}$ has been de-

fined, our algorithm runs the optimal on-line algorithm on $TDAMC^{approx}$.

B. Time Compression (TC)

In order to avoid running the optimal algorithm at every time-slot, our algorithm simply runs it every l_{TC} time-slots, and compares the expected energy consumption of $TDAMC(t)$ and $TDAMC(t+l_{TC})$ (instead of $TDAMC(t)$ and $TDAMC(t+1)$). Note that in this case, if the decision is made to upload, the uploading starts at time $t+l_{TC}$ with the channel in state $X_{t+l_{TC}}$, while the DP computations were done assuming that the current channel state is X_t. This is an additional source of approximation error for the algorithm.

C. Preemption Continuous Estimate (Preemption-CO)

In Preemptive offloading, the optimal algorithm DP is run over a $TDAMC$ whose states record also the remaining task bits to be uploaded, in order to account for all possible combinations of upoading time-slots, when it is estimating the expected energy consumption. In our algorithm, we propose that the *estimate* of expected energy consumption by the DP be done not on the *preemptive* $TDAMC$, but on the *1-Part* one, instead. Note that this algorithm should not be confused with the 1-Part optimal offloading algorithm; if the algorithm decides to upload at t, it uploads B_{X_t} bits (where B_{X_t} is the bit-rate at the channel state X_t), but continues to check whether to upload some of the remaining bits in the next time-slot, etc., as opposed to 1-Part offloading where once uploading is initiated, it continues automatically until it finishes.

IV. Simulation Results

In this section, computer simulation is used to evaluate the performance of the proposed approximation methods. A Gilbert-Elliot model, which is widely used for describing burst error patterns in transmission channels [4] [5] [7] [8] [9] [10] [11], is assumed for uploading. The channel is modelled as a two-state Markov chain with a "Good" (G) state having bit rate B_g and a "Bad" (B) state having bit rate B_b, and $B_g > B_b$. Simulations are conducted by applying the proposed approximation methods to the 1-Part Offloading and Preemption Offloading algorithms. For comparison, we also simulate Local Execution, in which the entire task is executed at the mobile device.

The default parameters used in the simulations are given as follows. All tasks are released at time-slot zero. Each time-slot is 1ms, and the transmit power is 1W, which results in the transmission energy in one time-slot as $E_{tr} = 1$mJ. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$mJ and the local computation power $f_l = 1$M CPU cycles per time-slot [12] [13]. The computation load of each task is $D = 10$M CPU cycles, and the local execution time is $T_L = D/f_l = 10$ time-slots. The local energy consumption $E_L = v_l D = 20$mJ. The data transmission rates are $B_g = 50$ Mbps and $B_b = 12.5$ Mbps. The channel state transition probabilities are $P_{GG} = 0.3$ (from G to G) and $P_{BB} = 0.7$ (from B to B). In the results below, each value of average energy consumption or running time is obtained by averaging 1000 random i.i.d. runs of the wireless channel.

1-Part-MC: By applying the proposed MC method on the 1-Part Offloading algorithm, we have 1-Part-MC. The compression factor is set as $l_{MC} = (\frac{S}{B_b})^{\frac{2-\alpha}{2}}$, where $\alpha \in [0, 2]$ with $\alpha = 0$ corresponding to the maximum compression and $\alpha = 2$ corresponding to no compression.

Figure 2a compares the running time of 1-Part-MC versus task size S with different α values. When $\alpha = 0$, the whole TDAMC is compressed into one node, and the algorithm can be run in a constant amount of time for any values of S. When $\alpha > 0$, the running time increases with both α and S, since the size of the approximated TDAMC increases with both.

Figures 2b and 2c, respectively, show the average energy consumption of the 1-Part-MC algorithm versus S and t_D with different α values. As α decreases (i.e., l_{MC} increases), more compression is done, resulting in less accurate offloading decisions that increase the average energy consumption of the mobile device. When α is small, the 1-Part-MC approaches the Local Execution faster as S increases (Figure 2b), and it requires larger t_D in order to achieve lower average energy consumption than the Local Execution (Figure 2c).

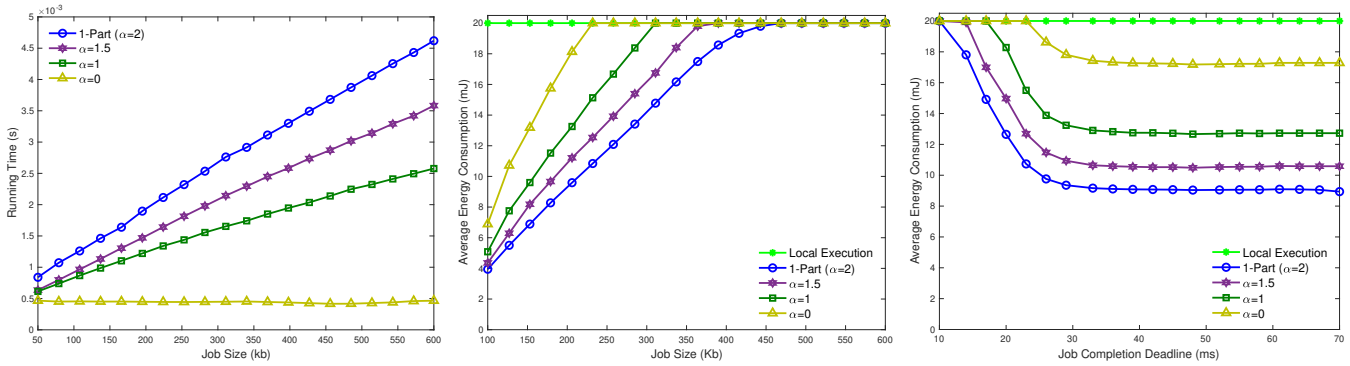
1-Part-TC: This is done by applying the TC method to 1-Part Offloading. The compression factor is $l_{TC} = {t_D}^{\frac{2-\beta}{2}}$, where $\beta = 0$, 1, 1.5, and 2. As a special case, when $\beta = 2$ ($l_{TC} = 1$), there is no compression.

Figure 3a shows the running time of the 1-Part-TC algorithm as S increases for different β values. By increasing l_{TC} (using smaller β) the running time of the algorithm can be reduced significantly. The compression factor is the largest when $\beta = 0$, in which case only one offloading decision is made for the whole task at the release time, and the running time keeps almost constant as S increases.

Figures 3b and 3c show the average energy consumption versus the task size S and deadline t_D, respectively. As the compression factor increases, the less accurate decisions result in higher energy consumption of the mobile device. When the compression factor is the largest ($\beta = 0$), the energy consumption of the mobile device is close to that of Local Execution. However, when $\beta = 1$ and 1.5, the compression causes a moderate increase in energy consumption but reduces significant reduction in the running time.

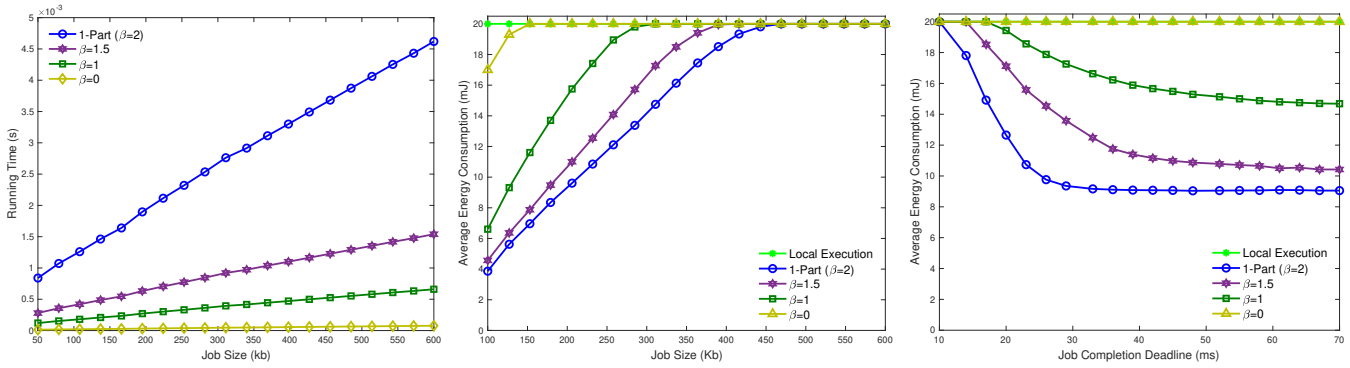
1-Part-MC-TC: In this set of simulations, the MC and TC methods are applied simultaneously to approximate 1-Part Offloading. Figure 4a shows that the running time can be more significantly reduced by using both the compression methods, compared to 1-Part-MC and 1-Part-TC. However, by applying both the methods at the same time, 1-Part-MC-TC suffers higher approximation errors and results in higher energy consumption as shown in Figures 4b and 4c.

Preemption-CO-MC: This is obtained by applying MC to the Preemption-CO 1-Part estimate (cf. Section III-C). The MC method is used to approximate the optimal expected offloading cost at each time-slot. Figure 5a shows the running time of Preemption-CO-MC as the task size changes for



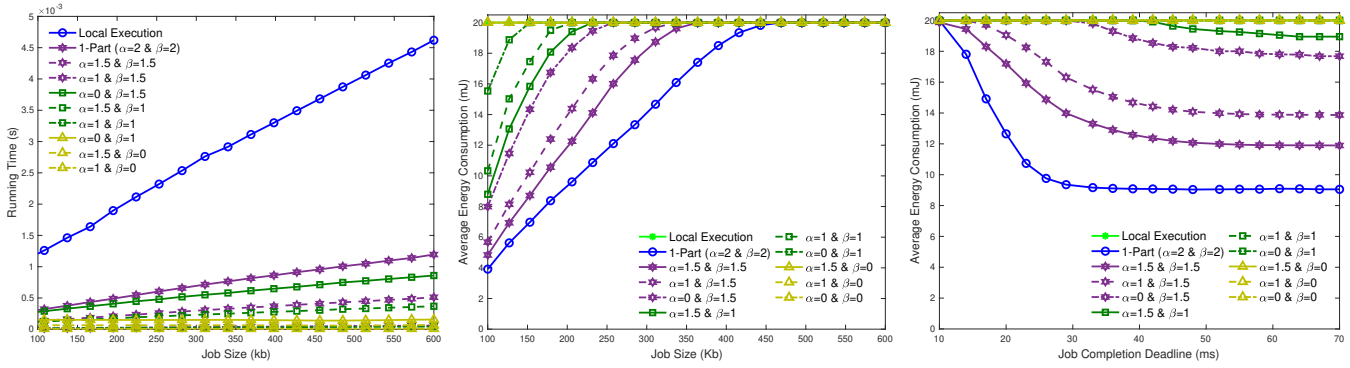
(a) t_D=60ms (b) t_D=60ms (c) $S = 200$kb

Figure 2: Approximation of 1-Part offloading using 1-Part-MC for different values of α



(a) t_D=60ms (b) t_D=60ms (c) $S = 200$kb

Figure 3: Approximation of 1-Part offloading using 1-Part-TC for different values of β



(a) t_D=60ms (b) t_D=60ms (c) $S = 200$kb

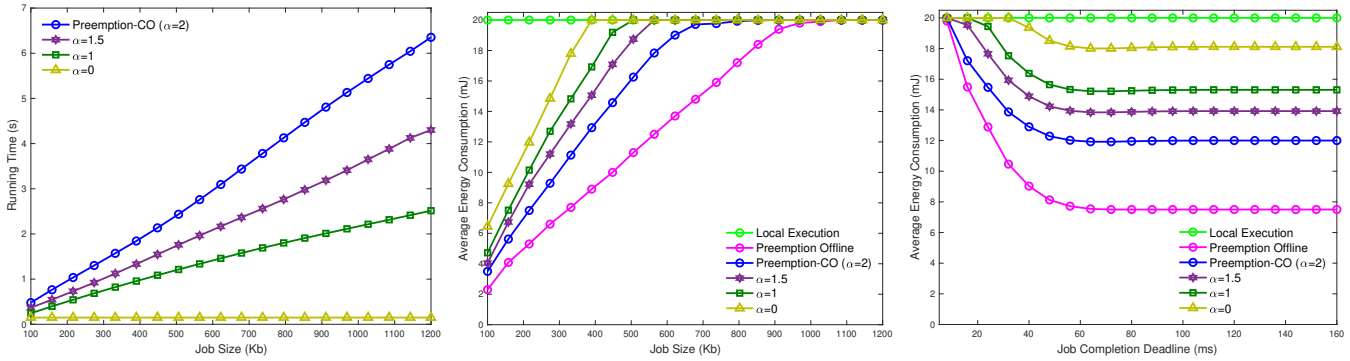
Figure 4: Approximation of 1-Part offloading using 1-Part-MC-TC for different combinations of α and β

different compression factors, where α is defined the same as in 1-Part-MC. By increasing the compression factor the running time can be reduced considerably. Figures 5b and 5c show the average energy consumption of the mobile device versus the task size and deadline, respectively. By decreasing α (increasing compression factor), the approximation error becomes larger, and the energy consumption increases. As a reference, we also plot Preemption Offline, which finds the optimal uploading times to offload the task preemptively by assuming a complete knowledge of all future channel states. The gap between Preemption-CO-MC with $\alpha = 2$ (no compression) and Preemption Offline is due to the fact that the former is ignorant of future channel states.

Preemption-CO-MC-TC: All three approximation algorithms in Section III are applied simultaneously to implement Preemption Offloading. Figure 6a shows that more significant reductions are achieved in running time, compared to Preemption-CO-MC. Figure 6b shows that the average energy consumption approaches Local Execution energy faster as S increases, and Figure 6c show that Preemption-CO-MC-TC requires a larger t_D to achieve lower average energy consumption than Local Execution.

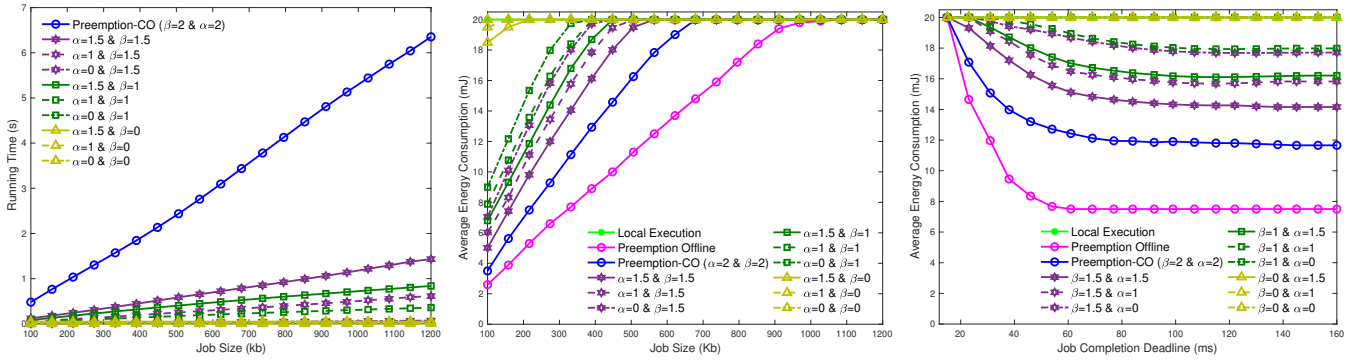
V. Conclusions

This paper has presented new algorithms and results that address the issue of algorithmic efficiency in mobile com-



(a) t_D=60ms (b) t_D=60ms (c) $S = 350$ kb

Figure 5: Approximation of Preemption offloading using Preemption-CO-MC for different values of α



(a) t_D=60ms (b) t_D=60ms (c) $S = 350$ kb

Figure 6: Approximation of Preemption offloading using Preemption-CO-MC-TC for different combinations of α and β

putation offloading. The case was considered where tasks to be run must always satisfy execution time deadlines. Hard deadlines are typically difficult to ensure in conventional computation offloading due to the stochastic nature of wireless channels used during the computation offload. Concurrent local execution (CLE) was used in order to ensure that all task execution deadlines are met. Offloading procedures have been formulated that can achieve the optimum minimum energy performance when the offloading occurs over stochastic Markovian wireless channels. The resulting algorithms, however, have a computational complexity that is too high for use in practical mobile implementations. The paper introduced three ways of reducing this complexity, namely, Markovian Compression (MC), Time Compression (TC) and Preemption Using Continuous Offloading (Preemption-CO). By applying these methods, the running times of the algorithms were shown to be significantly reduced without suffering significant performance loss.

REFERENCES

[1] Z. Gu, R. Takahashi, and Y. Fukazawa, "Real-time resources allocation framework for multi-task offloading in mobile cloud computing," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2019, pp. 1–5.

[2] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[3] H. A. Lagar-Cavilla, N. Tolia, E. De Lara, M. Satyanarayanan, and D. OHallaron, "Interactive resource-intensive applications made easy," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2007, pp. 143–163.

[4] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 14, no. 1, pp. 81–93, 2014.

[5] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.

[6] A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao, and G. Karakostas, "Optimal mobile computation offloading with hard deadline constraints," *IEEE Transactions on Mobile Computing*, 2019.

[7] T. Blazek and C. F. Mecklenbräuker, "Measurement-based burst-error performance modeling for cooperative intelligent transport systems," *IEEE Transactions on Intelligent Transportation Systems*, no. 99, pp. 1–10, 2018.

[8] A. Botta and A. Pescapé, "Ip packet interleaving for udp bursty losses," *Journal of Systems and Software*, vol. 109, pp. 177–191, 2015.

[9] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *The Bell System Technical Journal*, vol. 42, no. 5, pp. 1977–1997, 1963.

[10] M. Zafer and E. Modiano, "Minimum energy transmission over a wireless fading channel with packet deadlines," in *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 1148–1155.

[11] L. A. Johnston and V. Krishnamurthy, "Opportunistic file transfer over a fading channel: A pomdp search theory formulation with optimal threshold policies," *IEEE Transactions on Wireless Communications*, vol. 5, no. 2, pp. 394–405, 2006.

[12] M. Nir, A. Matrawy, and M. St-Hilaire, "An energy optimizing scheduler for mobile cloud computing environments," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 404–409.

[13] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *Wireless Communications, IEEE Transactions on*, vol. 11, no. 6, pp. 1991–1995, 2012.