Service Level Guarantee for Mobile Application Offloading in Presence of Wireless Channel Errors

Arani Bhattacharya*, Ansuman Banerjee†, Pradipta De*
*Department of Computer Science, SUNY Korea, {arani, pradipta.de}@sunykorea.ac.kr
†Indian Statistical Institute, {ansuman@isical.ac.in}

***Abstract*—Mobile cloud computing is increasingly being used in recent times to offload parts of an application to the cloud to reduce its finish time. However, quality of offloading decisions depend on network conditions and hence many offloading solutions assume that MAC layer retransmissions will tackle transient frame errors. This can lead to suboptimal solutions, as well as, degrade service level guarantee of reducing finish time compared to execution without offloading. In this work, we propose an error-aware solution that uses run-time channel conditions to adapt the offloading decisions. We guarantee that given a failure rate bound (ϵ), offloading decisions will achieve application execution in less time than that of local execution with a probability of (1-ϵ) while operating in networks with unpredictable error characteristics. Simulation results show that at channel error rate of 20%, our heuristic provides 90% guarantee of better performance than on-device computation and reduces the mean finish time by 18% compared to execution without any offloading.**

***Index Terms*—Mobile Cloud, Application Offloading, Cross-Layer Network Optimization**

I. Introduction

Mobile computing platforms, from smart sensors to smartphones, are increasingly used in personal and enterprise environments. However, these devices have limited compute capacity. This limitation can be mitigated by offloading parts of a mobile application to execute on cloud servers, thereby reducing application finish time. A number of proposals [1], [2] for mobile cloud computing have received prominence in literature. Among other factors, offloading decisions depend on network conditions. Since network conditions in mobile environment vary widely, offloading decisions based on profiled network parameters can lead to sub-optimal solutions.

Channel error rates are one of the hardest to model among network parameters. Channel error is dependent on unpredictable external interference, and mobility characteristics, like walking or driving. Measurement based studies have shown channel error rates up to 30% under different conditions [3], [4]. Therefore, offloading solutions depend on the MAC layer retransmission mechanism to handle channel errors. Since the number of retransmissions can depend on transient channel error states, this can undermine the benefit of offloading in saving energy and/or finish time.

We illustrate with an example. We take a task graph with 100 tasks, where a task, representing a method in the application execution, can be offloaded to the remote cloud server for faster execution. Given each task's workload profile, and network parameters, an optimization solver (as in MAUI [1]) computes the *estimated* time to finish execution. Fig. 1 shows a comparison of application finish time using four schemes: *Estimated* is the result of using an optimization solver with application and network profile as input, *local* computes without any offloading, *actual* is the result of offloading in practice due to channel errors, and an hypothetical solution (called oracle in the figure) assumes complete knowledge of channel errors. Compared to *local* execution, where no task is offloaded, an offloading scheme performs better. However, in practice, the channel error conditions can break the assumption about network parameters. In presence of varying channel errors, the *actual* result of offloading may not be as computed by an optimization solver. An *oracle* solution, with complete knowledge of channel errors, can indeed perform better.

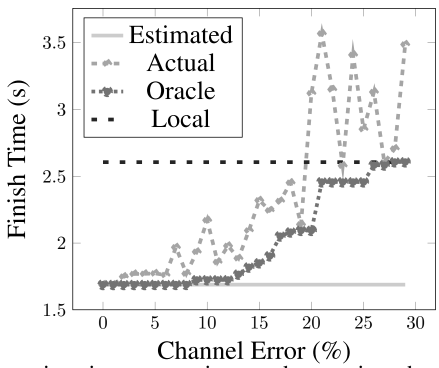

Fig. 1: Execution time comparison under varying channel errors.

In this work, we pose the question, *even in presence of unpredictable channel errors, can we ensure service level guarantee to complete the application execution faster than that of local execution on the device?* We show that, given a failure rate bound, the question can be modeled as a chance constrained optimization problem [5]. We propose an error-aware run-time adaptive heuristic that decides at each task offload point, the locally optimal choice considering stochastic channel errors. We provide guarantee to minimize the expected application finish time. Our scheme ensures that an application completes execution faster than local execution, in presence of retransmissions due to channel errors. We validate our solution using simulations and on traces of benchmark applications.

We present relevant prior work in Section II. Section III and Section IV present the analytical model and the proposed heuristic. Evaluation is presented in Section V followed by conclusion in Section VI.

$\mathbb{V}$	Vertex set of the graph
$\mathbb{E}$	Edge set of the graph
v_j	A task in the application execution
v_m	Last task in the graph
(v_i, v_j)	A dependency from the task v_i to v_j
$\mathcal{M}_0$	Mobile device
$\mathcal{M}_1$	Cloud server
t_j^l	Execution time of task v_j on machine $\mathcal{M}_l$
r	Time to migrate a single frame
U_m	Time deadline given to application
ϵ	Failure bound given to application
w_{ij}	Number of frames needed to migrate (v_i, v_j)
x_j	Variable indicating execution of v_j on $\mathcal{M}_0$ or $\mathcal{M}_1$
z_{ij}	Maximum number of retransmission attempts of (v_i, v_j)
Y_{ij}	Number of retransmission attempts of frames of (v_i, v_j)
R_{ij}	Total time to migrate (v_i, v_j)
T_j	Finish time of v_j
α_k	Failure bound on k^{th} migration
α_k^s	Failure bound on sending packet of k^{th} migration
α_k^r	Failure bound on receiving packet of k^{th} migration

TABLE I: Symbols introduced in Section III

II. Related Work

There are two different categories of work in the context of offloading over wireless channels. One group of work assumes that the Medium Access Control (MAC) layer handles channel errors successfully. The first offloading frameworks, MAUI [1] and CloneCloud [2], used this approach. They estimated the channel bandwidth before solving the offloading decision problem. Another offloading framework, ThinkAir [6] looks at history of migration and assumes that the channel conditions remain similar to the past observations. Some other works try to reduce the amount of data migration. [7] proposes compiler-level optimizations to decide which data is actually used by the cloud server. These offloading frameworks do not consider the cost of transmission failure.

Finally, a few studies have considered the effect of channel errors. [8] shows how intelligent checkpointing of applications to ensure consistency on the mobile device and the cloud server can save energy of offloaded applications. COSMOS [9] senses the response time to determine the quality of connection, and uses this observation for the offloading decision. However, they do not consider retransmission of lost packets. In [10], the authors consider retransmission, but the decision about the number of retransmissions is not made at run-time. In Foreseer [11], the initial partition obtained by running an optimization solver is modified at run-time based on the channel bandwidth. In contrast to the work above, our proposal models the number of retransmissions due to channel errors and presents an adaptive offloading algorithm design.

III. Models and Problem Formulation

We represent execution of a mobile application as a directed acyclic graph (DAG) $G = (\mathbb{V}, \mathbb{E})$, where the vertex set $\mathbb{V}$ represents the set of m methods or tasks, and the edge set $\mathbb{E}$ represents the dependencies among tasks. A task can be executed either locally on the mobile device, $\mathcal{M}_0$, or on the remote cloud server, $\mathcal{M}_1$. However, the first and last task, v_1 and v_m respectively, must execute on the mobile device. If a task v_j is executed on a platform, $\mathcal{M}_0$ or $\mathcal{M}_1$, different from that of any of its predecessor tasks, v_i's, where $(v_i, v_j) \in \mathbb{E}$, then the task output states of v_i must be transferred over the network to v_j's execution platform. Since the data transfer size will vary across dependencies, therefore, the number of data fragments or frames at the MAC layer will also vary.

The wireless channel is modeled as a stochastic process [12], where the probability of successful transmission of a frame is denoted by p. The value of p depends on the time varying nature of the channel. However, we assume that for a single data packet (i.e. for all the corresponding frames) the channel state remains unchanged. Due to channel errors, if a frame is lost, it is retransmitted. Let Y_{ij} be the total transmission attempts for all the frames of a packet transferring data from v_i to v_j. If the time to transmit a frame is r, then the time, R_{ij}, for the packet transmission will therefore be,

$$R_{ij} = rY_{ij}$$

Since Y_{ij} depends on the channel conditions, both Y_{ij} and R_{ij} are stochastic parameters.

The total time to execute an application depends on where each task is executed (i.e. execution time) and the time for the network transfer (i.e. migration time). Note that time to execute an application is same as the finish time, T_m, of the last task, v_m. T_m depends on the time for network transfers (R_{ij}'s), and is therefore also a stochastic parameter. Let U_j denote the time taken to finish v_j if v_j and all tasks preceding it are executed locally. Our objective is to minimize the expected finish time, T_m, under a constraint that T_m exceeds the local execution time, U_m only with a fixed probability ϵ. The constraint guarantees a service level agreement (SLA) that the application finish time will exceed local execution time (U_m) with maximum probability ϵ while offloading to cloud in unpredictable channel conditions. We express this as a chance constrained optimization problem:

$$\begin{aligned} &\textbf{Min } E[T_m] \\ &\textbf{subject to: } \mathbb{P}(T_m > U_m) \leq \epsilon \end{aligned} \tag{1}$$

We now explain the nature of this optimization problem. Since there are some tasks in the DAG that must be executed on $\mathcal{M}_0$, there may be multiple send and receive migrations to the cloud server. We consider these migrations in pairs. A send migration offloads the data needed by an offloaded method from $\mathcal{M}_0$ to $\mathcal{M}_1$, while a receive migration sends data back from $\mathcal{M}_1$ to $\mathcal{M}_0$. Corresponding to every send migration of a method, we can therefore uniquely associate a receive migration of another method before the next send migration is initiated. We leverage on this pairwise send-receive association to build the foundation of our theory. In our work, for the sake of simplicity, we use a migration to denote a send-receive association pair. Then, we define the event of failure of a single migration as "execution time greater than local execution time". We denote the failure for k^{th} migration attempt as F_k, i.e. F_k is true if $(T_j > U_j)$ where v_j is executed on mobile device. Since the condition of the channel may change between migrations, it is possible that after a single migration is completed, the channel condition degrades to allow no further migrations. Thus, failure of a single migration may lead to failure of the entire execution. We therefore, rewrite the chance constraint as:

$$\mathbb{P}(\bigcup_k F_k) \leq \epsilon, \tag{2}$$

where k varies over the number of migrations during the application's execution from start to finish. Using inclusion-exclusion principle [13], we rewrite this as:

$$\sum_k \mathbb{P}(F_k) \leq \epsilon \tag{3}$$

A conservative way to satisfy Eqn 3 is by imposing a failure bound α_k on each migration:

$$\mathbb{P}(F_k) \leq \alpha_k, \quad \forall k \text{ such that: } \sum_k \alpha_k \leq \epsilon \tag{4}$$

As before, a single migration consists of two different probabilistic events: sending a packet to cloud and receiving it back to mobile device. Then, the total time available for migration to satisfy deadline may be divided up into three components: sending a packet, executing tasks on cloud and receiving a packet. Since only sending and receiving are probabilistic events, we define F_k^s and F_k^r as failure while sending and receiving respectively. Here, F_k^s and F_k^r are defined as events denoting failure to send and receive a packet within an assigned time (to be detailed in the following) that guarantees SLA satisfaction. As in Eqn 4, we bound the probability of failure while sending and receiving by α_k^s and α_k^r respectively:

$$\mathbb{P}(F_k^s) \leq \alpha_k^s \text{ and } \mathbb{P}(F_k^r) \leq \alpha_k^r \text{ such that: } \alpha_k^s + \alpha_k^r = \alpha_k \tag{5}$$

We need α_k^s and α_k^r that minimizes the overall application finish time. We now establish a bound on the number of transmission attempts for each individual send or receive migration. Let z_{ij} be the maximum number of transmission attempts for a send or receive migration between v_i and v_j. The values of α_k^s and α_k^r determine the value of z_{ij}. We assume a single packet of (v_i, v_j) data contains w_{ij} frames. Thus, if migration (either send or receive) is performed, the actual number of transmission attempts Y_{ij} must satisfy:

$$w_{ij} \leq Y_{ij} \leq z_{ij} \tag{6}$$

We need to find values of z_{ij} that minimize the overall execution time while satisfying to satisfy SLA. Increasing z_{ij} reduces the failure rate. However, this also increases the expected application finish time.

IV. Solution Approach

In this section, we design a heuristic that minimizes application finish time. We denote z_{ij}^s and z_{ij}^r as the maximum number of transmission attempts for send and receive migrations respectively. This requires allowing a maximum z_{ij}^s and z_{ij}^r transmission attempts while sending and receiving packets from cloud server.

We explain our methodology on z_{ij}^s. The computation of z_{ij}^r is similar. Sending a (v_i, v_j) packet succeeds only if all of its w_{ij} frames are successfully transmitted. Let Q_{ij} be a random variable denoting the number of frames successfully transmitted in a total of z_{ij}^s transmission attempts. Then, failure to send a dependency to the cloud server (F_k^s) occurs when less than w_{ij} frames are transmitted successfully in z_{ij}^s transmission attempts. We, therefore, rewrite Eqn 5 as follows:

$$\mathbb{P}(Q_{ij} < w_{ij}) \leq \alpha_k^s \tag{7}$$

As discussed before in our channel model, the probability p of successful transmission remains same while sending frames of a single packet. Thus, we can treat Q_{ij} as a binomial random variable with the parameters z_{ij}^s and p, i.e. $Q_{ij} \sim Binomial(z_{ij}^s, p)$. There is no closed form formula to find the probability of success of at least w_{ij} trials in z_{ij}^s attempts [14]. We, therefore, find an approximate value of z_{ij}^s using Hoeffding's inequality [15]. Hoeffding's inequality states that for a random variable, $Q_{ij} \sim Binomial(z_{ij}^s, p)$, the deviation from the mean t (where $t < 0$) is bounded by:

$$\mathbb{P}(Q_{ij} - E[Q_{ij}] \leq t) \leq \exp\{-2t^2/z_{ij}^s\} \tag{8}$$

We rewrite Eqn 7 as shown below to match Eqn 8.

$$\begin{aligned}
&\mathbb{P}(Q_{ij} < w_{ij}) \leq \alpha_k^s \\
\implies &\mathbb{P}(Q_{ij} - z_{ij}^s p < w_{ij} - z_{ij}^s p) \leq \alpha_k^s \\
\implies &\mathbb{P}(Q_{ij} - E[Q_{ij}] < w_{ij} - z_{ij}^s p) \leq \alpha_k^s \\
\implies &\mathbb{P}(Q_{ij} - E[Q_{ij}] \leq w_{ij} - z_{ij}^s p - 1) \leq \alpha_k^s \\
\implies &\exp\{\frac{-2(w_{ij} - z_{ij}^s p - 1)}{z_{ij}^s}\} \leq \alpha_k^s
\end{aligned} \tag{9}$$

Taking logarithm of both sides of Eqn 9, and solving for z_{ij}^s gives us the solution:

$$z_{ij}^s \geq \frac{4w_{ij}p - 4p - \ln(\alpha_k^s) + \sqrt{(4w_{ij}p - 4p - \ln(\alpha_k^s))^2 + 8p^2(w_{ij}-1)^2}}{4p^2}$$

z_{ij}^s represents the minimum number of send attempts needed to satisfy the SLA. Since increasing the number of transmission attempts also satisfy the SLA, we can utilize the inequality $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ in the above expression for z_{ij}^s to get a higher bound on z_{ij}^s as:

$$z_{ij}^s \geq \frac{8w_{ij}p - 8p - 2\ln(\alpha_k^s) + 2\sqrt{2}p(w_{ij} - 1)}{4p^2} \tag{10}$$

Eqn 10 expresses the SLA constraint for sending (Eqn 5) in terms of number of transmission attempts z_{ij}^s. z_{ij}^s being an integer, we write z_{ij}^s as:

$$\begin{aligned}
z_{ij}^s &= \lceil \frac{8w_{ij}p - 8p - 2\ln(\alpha_k^s) + 2\sqrt{2}p(w_{ij} - 1)}{4p^2} \rceil \\
&= \lceil \frac{p(w_{ij} - 1)(4 + \sqrt{2}) - \ln(\alpha_k^s)}{2p^2} \rceil
\end{aligned} \tag{11}$$

As discussed earlier, the k^{th} migration also involves receiving a packet of $(v_{i'}, v_{j'})$ from cloud server to mobile device. Solving the SLA constraint involves finding both z_{ij}^s and $z_{i'j'}^r$.

Using the same method that we used for z_{ij}^s, we find the number of transmissions $z_{i'j'}^r$ to receive a packet:

$$z_{i'j'}^r = \lceil \frac{p(w_{i'j'} - 1)(4 + \sqrt{2}) - \ln(\alpha_k^r)}{2p^2} \rceil \tag{12}$$

So far, we have the values of z_{ij}^s and $z_{i'j'}^r$ in terms of weight parameters α_k^s and α_k^r respectively. We need to find values of α_k^s and α_k^r that minimize total time to send and receive packets, i.e. network time. We note that the time to send and receive a packet is equal to $z_{ij}^s \times r$ and $z_{i'j'}^r \times r$ respectively. Thus, total network time is given by $z_{ij}^s \times r + z_{i'j'}^r \times r$. We differentiate this with respect to α_k^s and set the derivative to 0 to obtain $\alpha_k^s = \alpha_k^r = \alpha_k/2$. Therefore, we replace α_k^s and α_k^r in the expressions of z_{ij}^s and $z_{i'j'}^r$ respectively by $\alpha_k/2$:

$$z_{ij}^s = \lceil \frac{p(w_{ij} - 1)(4 + \sqrt{2}) - \ln(\alpha_k/2)}{2p^2} \rceil \tag{13}$$

$$z_{i'j'}^r = \lceil \frac{p(w_{i'j'} - 1)(4 + \sqrt{2}) - \ln(\alpha_k/2)}{2p^2} \rceil \tag{14}$$

The above gives us the values of z_{ij}^s and $z_{i'j'}^r$ needed to satisfy SLA in terms of α_k for the different migrated edges.

We now need to assign values of α_k for each migration. The values of α_k must be assigned in a way that satisfies Eqn 5. Moreover, the total number of possible migrations are not known. A conservative strategy is to choose higher values of α_k for the early migrations, since saving time at the beginning increases the time available for later migrations. Thus, we choose α_k as a geometric distribution, with a ratio of $1/2$ as shown below:

$$\alpha_k = \frac{\epsilon}{2^k} \tag{15}$$

Our heuristic now follows directly from this calculation. It takes as input the set of tasks $\mathbb{V}$, the set of tasks $\mathbb{E}$, the time deadline U_m, failure bound ϵ and time to transmit a single frame r. It then executes each task either on mobile device or cloud server. Whenever a task v_j is ready for execution on the mobile device ($\mathcal{M}_0$) or the cloud server ($\mathcal{M}_1$), we check whether executing it on the same machine or migrating it saves time. The time required for migration is obtained by sensing the channel condition at each step to find the probability p of successful transmission and using it to calculate the number of transmission attempts z_{ij}^s and $z_{i'j'}^r$. For simplicity, since z_{ij}^s and $z_{i'j'}^r$ have the same expressions, we refer to it as z_{ij} in our heuristic. If migration is faster, then a packet of (v_i, v_j) is migrated. While migrating, sending of a packet from mobile device to cloud server can be aborted before transmitting all frames if the number of failures is high. However, this is not possible for receiving a packet from cloud server to mobile device, since execution must finish on mobile device. The exact algorithm is shown in detail in Algorithm 1.

We now analyze the time complexity of our method. The Procedure CALCULATE-BUDGET iterates over all dependencies in the application. Thus, it has a time complexity of $O(|\mathbb{E}|)$. Procedure EXECUTE-APPLICATION iterates over each task in the graph. For each task, it calls CALCULATE-BUDGET once. Thus, the total complexity of computing the overall budget is $O(|\mathbb{V}||\mathbb{E}|)$ It also has an inner loop that iterates over each dependency of a single task. Assuming the number of frames to be transmitted as constant, this has a time complexity of $O(|\mathbb{E}|)$. Therefore, total time complexity of using our algorithm is equal to $O(|\mathbb{V}||\mathbb{E}|)$. Assuming a constant number of parallel tasks, and since $|\mathbb{V}| = m$, the time complexity is equal to $O(m^2)$.

Algorithm 1 Our channel error offloading algorithm.

procedure EXECUTE-APPLICATION($\mathbb{V}$, $\mathbb{E}$, U_m, ϵ, r)
  x[1] $\leftarrow 0$
  $k \leftarrow 1$
  Execute first task v_1 on mobile device
  for all $v_j \in \mathbb{V}$ ready for execution **do**
    Get the probability of successful transmission p
    $\alpha_k = \epsilon/2^k$
    CALCULATE-BUDGET($\mathbb{V}$, $\mathbb{E}$, U_m, p, α_k, r)
    Y $\leftarrow 0$
    for all $(v_i, v_j) \in \mathbb{E}$ **do**
      Calculate number of frames w_{ij} for migration
      $z_{ij} = \lceil \frac{p(w_{ij}-1)(4+\sqrt{2})-\ln(\alpha_k/2)}{2p^2} \rceil$
      if x[i] = 0 & mobBudget[j] $>$ cldBudget[j] $+z_{ij}r$ **then**
        migTime $\leftarrow T_i$ - cldBudget[j]
        x[j] $\leftarrow 1$
        f $\leftarrow 1$
        while $f \leq w_{ij}$ & x[j] = 1 **do**
          maxAttempts $\leftarrow$ migTime / rw_{ij}
          Attempt migration of f^{th} frame maxAttempt times
          if migration of frame failed **then**
            x[j] $\leftarrow 0$
          end if
          Store number of frame transmission attempts in Y_{ij}
          f $\leftarrow$ f + 1
        end while
      else if x[i] = 0 & mobBudget[j] $\leq$ cldBudget[j]$+z_{ij}r$ **then**
        x[j] $\leftarrow 0$
      else if x[i] = 1 & mobBudget[j] $+z_{ij}r >$ cldBudget[j] **then**
        Attempt transmission of frames till successful migration
        Store number of frame transmission attempts in Y_{ij}
        x[j] $\leftarrow 0$
        $k = k + 1$
      else if x[i] = 1 & mobBudget[j] $\leq$ cldBudget[j] $+z_{ij}r$ **then**
        x[j] $\leftarrow 1$
      end if
    end for
    $Y \leftarrow \max(Y, Y_{ij})$
    $h \leftarrow$ x[j]
    Execute v_j on $\mathcal{M}_h$
    $T_j \leftarrow T_i + rY + t_j^h$
  end for
end procedure
procedure CALCULATE-BUDGET($\mathbb{V}$, $\mathbb{E}$, U_m, p, α_k, r)
  cldBudget[m] $\leftarrow \infty$
  mobBudget[m] $\leftarrow U_m - t_m^0$
  $C \leftarrow \{v_m\}$
  for all $v_j \in C$ **do**
    for all $(v_i, v_j) \in \mathbb{E}$ **do**
      Let w_{ij} be number of frames to migrate (v_i, v_j)
      $z_{ij} \leftarrow \lceil \frac{p(w_{ij}-1)(4+\sqrt{2})-\ln(\alpha_k/2)}{2p^2} \rceil$
      mobTime[j] $\leftarrow$ max(mobBudget[j] - t_i^0,cldBudget[j] - t_i^1 - z_{ij}r)
      cldTime[j] $\leftarrow$ max(cldBudget[j] - t_i^1, mobBudget[j] - t_i^0 - z_{ij}r)
      mobBudget[i] $\leftarrow$ min(mobBudget[i], mobTime[j])
      cldBudget[i] $\leftarrow$ min(cldBudget[i], cldTime[j])
    end for
    $C \leftarrow C \cup v_i$
  end for
end procedure

V. EVALUATION

In this section, we evaluate the performance of our algorithm using simulation on both randomly generated graphs and benchmark programs.

Parameter	Range of Values
Migration time of each packet (r)	50 ms
Server speed compared to mobile device	5 times
Processor power	1 J/s
Network power	0.5 J/s
Number of random graphs	10000
Failure bound	1%
Channel error rate	30%

TABLE II: Parameters used for each simulation experiment. Unless mentioned otherwise, these parameters are used in the experiments.

A. Settings

We implement our heuristic at different channel error rates and failure bounds. To better understand the performance of our algorithm, we implement an Integer Linear Programming (ILP) based solution which assumes that there is no channel error. We also implement another ILP-based solution called oracle which knows in advance the cases in which transmission attempts fail. We have assumed in our simulation that the channel error rate varies around the mean with uniform distribution. The simulation parameters are given in Table II.

B. Simulation Results

To study the performance of our heuristic, we first run the ILP-based solution, oracle and our heuristic on a set of 10000 randomly generated graphs. We then compare the failure rate, mean finish time and energy consumption of our heuristic with the ILP-based solution and the oracle.

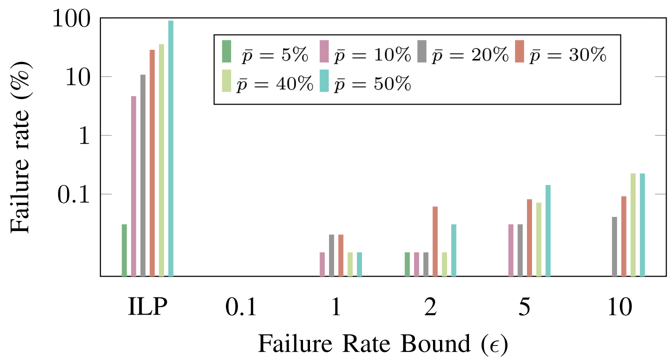

Fig. 2: Comparison of failure rate at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). Failure represents a finish time higher than local execution.

1) Failure rate: We compare the failure rates of the three implementations to check whether our algorithm satisfies the failure bound. Fig. 2 shows the failure rates under different channel conditions compared to ILP based solution. We omit the oracle implementation since it knows in advance the cases of transmission failure and therefore, can never fail. We also do not show channel error rate of 2%, since the number of failures at 2% is too small. At channel error rates of 5%, 10% and 30%, the ILP gives a failure rate of 0.03%, 4.5% and 28.1% respectively. The failure rates for our solution are bounded within 10% even at 30% channel error, giving a service level guarantee of 90%. The number of failures in our scheme never exceeds the defined failure bound ϵ.

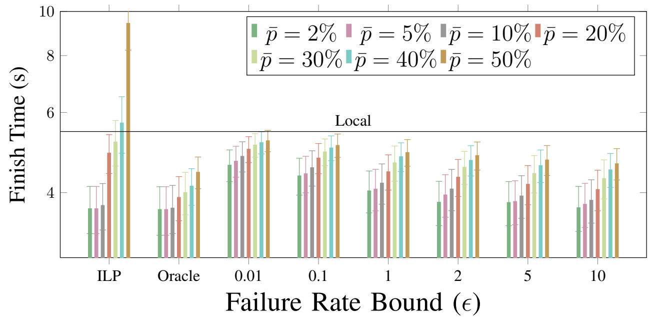

Fig. 3: Comparison of finish time at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). Oracle solution represents best possible finish time for a given level of channel error.

ϵ \ $\bar{p}$	5	10	20	30	40	50
0.1	22	19	14	10	8	7
1	31	28	22	17	14	11
2	34	32	25	20	16	13
5	39	36	30	24	20	16
10	41	39	34	28	23	19

TABLE III: Percentage of tasks executed on cloud server at different channel error rates ($\bar{p}$) and failure bounds (ϵ).

These observations confirm that since ILP runs a priori, its solution might lead to worse than expected results while executing the application. Although our heuristic does not guarantee an optimal solution, it can sense the channel condition and decide accordingly whether to offload. This reduces the number of failures compared to an ILP. Moreover, when the number of errors in the wireless channel increases, our heuristic reduces the chances of failure by offloading tasks to the cloud. We confirm this observation by noting in Table III that the number of tasks executed on cloud server decreases with a decrease in failure bound (ϵ).

We also note that in a few cases the number of failures decreases with an increase in channel error. However, this decrease in failure at a higher channel error rate is less than 0.2%, which may be explained by the uncertain nature of the wireless network.

2) Finish Time: We compare the finish times of our heuristic with the ILP-based and oracle solutions. Fig. 3 shows the mean finish time of the application samples under varying channel error rates. The heuristic has a better average performance than global optimization solver for channel error rate greater than 10%. When the channel error exceeds 20%, our heuristic takes less time than the ILP solution in all cases, with a failure rate of 10% giving a gain of 18%. Below 20% error, our heuristic provides a solution within 5% of the ILP solution for all values of ϵ. At error rate of 50%, the ILP takes twice the finish time of our heuristic.

We explain these observations by noting that an ILP obtains the best possible solution when there is no channel error. Thus at lower levels of channel error, it performs better, because channel error does not lower finish time significantly. When the number of channel errors increases, our heuristic performs better since it is able to adapt to the channel condition.

3) Energy Consumption: We now investigate the effect of our heuristic on energy consumption of the battery in the mobile device. Since a mobile device runs on battery, reducing usage of battery energy is important for mobile users. We assume that execution on mobile device consumes power of 1 J/s, while network transmission takes 0.5 J/s. Fig. 4 compares the energy consumption of our heuristic with the ILP based solution. We note that energy consumption follows the same trend as finish time. This is because, the power consumption of processor system is greater than the network card. Thus, reducing the number of tasks that are executed on mobile device also reduces its energy consumption.

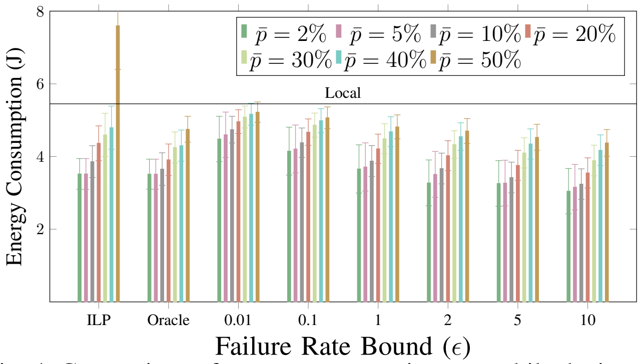

Fig. 4: Comparison of energy consumption on mobile device at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). We have obtained the energy consumption by assuming that processor power = 1 J/s and network power = 0.5 J/s.

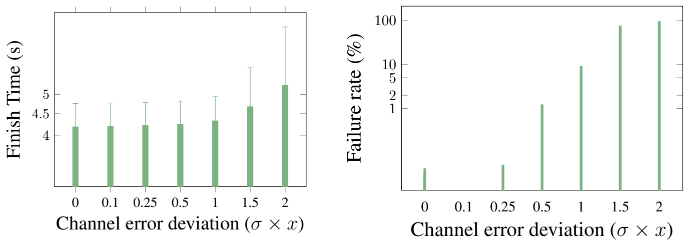

Fig. 5: Effect of deviation in channel condition when tasks are executed on cloud server on finish time and failure rate. The amount of deviation is measured in multiples of the channel standard deviation.

4) Effect of Channel Variation: We study how deviation in the channel error rate during execution on cloud server affects the performance of our heuristic. Fig. 5 shows the impact of change in the channel condition during execution on cloud server on failure rate and finish time. We note that the mean finish time is not affected much by changes in the channel condition. However, both the standard deviation of the finish time and the failure rate increases when the channel condition changes more frequently.

We explain these observations by noting that the condition of the channel may either improve or worsen. The overall effect, therefore, balances out to give a mean finish time close to the solution given by the heuristic. However, the cases in which the channel has more errors result in failure. Thus, the number of failures increases with an increase in deviation of the channel error rate. This also explains the increase in the standard deviation of the application finish times when channel error increases.

C. Trace-driven Results

To further confirm that our results are practical, we generate graphs from execution traces of SPECjvm08 benchmarks. We utilized AspectJ framework to generate traces of six SPEC benchmark programs: compress, scimark.monte-carlo, crypto.aes, mpegaudio, scimark.fft.small and cypto.rsa. These benchmarks were chosen based on the workloads that are most commonly run on mobile devices. Of these benchmarks, the programs compress, scimark.monte-carlo and crypto.aes are compute-intensive. The other programs mpegaudio, crypto.aes and crypto.rsa are input-intensive as they read data from a file.

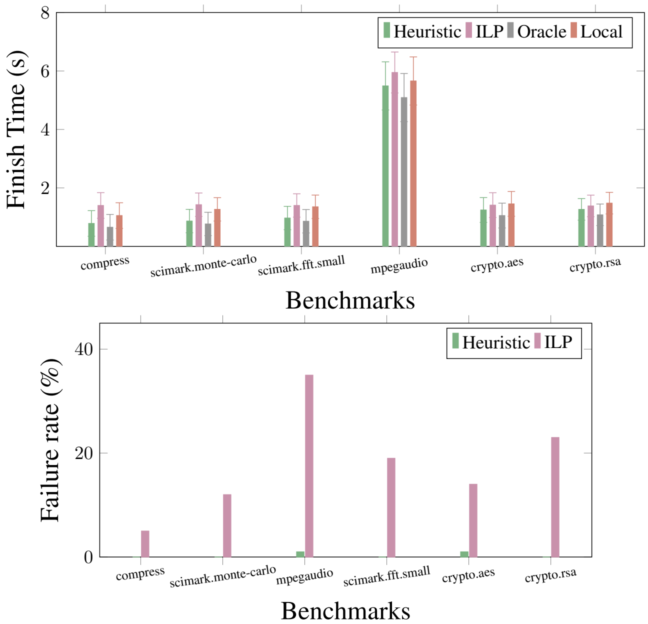

Fig. 6: Comparison of finish time and failure rate of six different SPECjvm2008 benchmarks using execution on mobile device (local), oracle solution, our heuristic and ILP. Each benchmark has been executed 100 times.

Fig. 6 show the finish time and failure rate on each of these benchmark programs. We note that in each case, the finish time is lower than the ILP, but higher than the oracle solution. This confirms our finding that our heuristic gives a better finish time in the presence of channel errors. Moreover, for the input-intensive applications, the ILP solution has a higher finish time than local execution. From the failure plot, we also note that the failure rate is lower than 1% for each of the benchmark programs. This is much lower than the ILP solution, where the failure rates are all higher than 10%.

These observations confirm that our adaptive heuristic works on realistic workloads. Moreover, input-intensive applications require higher number of migrations, and thus lead to more failures using an ILP-based solution. Our heuristic can reduce failure while executing input-intensive applications by reducing the number of tasks executed on cloud server when the channel error probability is high.

VI. Conclusion

Offloading of mobile applications to cloud servers can augment the limited compute capacity of their processors. However, the quality of offloading based execution depends on the network parameters, like channel error conditions. Unbounded retransmissions to handle channel errors can lead to service degradation as it may end up taking longer than local execution time to complete the application. In this work, we propose an adaptive algorithm that tracks the channel error, defines a stochastic model to capture channel conditions, and uses it to adjust the number of retransmissions to deliver a better service level guarantee in completing an application compared to optimization solutions. The mean finish time of an application is also comparable to typical solutions. We show the efficacy of our technique on both traces and randomly generated application profiles.

Our study has a few limitations. First, we assume that the channel error during a single migration remains same. This may not hold true in a rapidly varying channel. However, we have shown through simulation that a rapidly varying channel affects finish time only when the amount of channel variation is high. Secondly, our algorithm does not guarantee the minimum possible expected finish time. We provide a heuristic that reduces the application finish time compared to local execution under different channel conditions.

References

[1] E. Cuervo, A. Balasubramanian, D.-k. 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.

[2] B.-G. 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*. ACM, 2011.

[3] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, “Predictable 802.11 packet delivery from wireless channel measurements,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 159–170, 2011.

[4] J. Gozalvez, M. Sepulcre, and R. Bauza, “Impact of the radio channel modelling on the performance of vanet communication protocols,” *Telecommunication Systems*, vol. 50, no. 3, pp. 149–167, 2012.

[5] E. Erdoğan and G. Iyengar, “Ambiguous chance constrained problems and robust optimization,” *Mathematical Programming*, vol. 107, no. 1-2, pp. 37–61, 2006.

[6] 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 *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012.

[7] S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, and Y. Paek, “Fast dynamic execution offloading for efficient mobile cloud computing,” in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*. IEEE, 2013, pp. 20–28.

[8] Y.-W. Kwon and E. Tilevich, “Energy-efficient and fault-tolerant distributed mobile execution,” in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 2012, pp. 586–595.

[9] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, “Cosmos: computation offloading as a service for mobile devices,” in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2014, pp. 287–296.

[10] W. Zhang, Y. Wen, and D. Wu, “Collaborative task execution in mobile cloud computing under a stochastic wireless channel,” vol. 14, no. 1, Jan 2015, pp. 81–93.

[11] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri, “Run time application repartitioning in dynamic mobile cloud environments,” *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[12] M. Patzold, *Mobile fading channels*. John Wiley & Sons, Inc., 2003.

[13] W. Szpankowski, “Inclusion-exclusion principle,” *Average Case Analysis of Algorithms on Sequences*, pp. 49–72, 2001.

[14] R. W. Gosper, “Decision procedure for indefinite hypergeometric summation,” *Proceedings of the National Academy of Sciences*, vol. 75, no. 1, pp. 40–42, 1978.

[15] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.