A framework for speculative scheduling and device selection for task execution on a mobile cloud

A. BANERJEE, Indian Statistical Institute Kolkata, India
H. S. PAUL, A. MUKHERJEE, S. DEY and P. DATTA, Innovation Labs, TCS, Kolkata, India

In this paper, we study the problem of opportunistic task scheduling and workload management in a mobile cloud setting considering computation power variation. We gathered mobile usage data for a number of persons and applied supervised clustering to show that a pattern of usage exists and that follows a state-based model. Based on this model, we present a strategy to choose and offload work on a mobile device. We present a framework and experimental results showing the efficacy of our proposed approach.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**]: Distributed applications—*Mobile Computing*

Additional Key Words and Phrases: Mobile Grid Computing, Task Scheduling, Usage Pattern

ACM Reference Format:

A Banerjee et al. 2014. A framework for speculative scheduling and device selection for task execution on a mobile cloud *jn* 0, 0, Article 0 (July 2014), 11 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The growing market for smart devices is stimulating the prospect of utilizing them as computing resources [Bonomi et al. 2012; Mukherjee et al. 2014]. Recent studies on the computing capacity of mobile devices claim that their computing power is comparable to that of desktops [Sakr 2011]. This has led to several proposals of Mobile Cloud Computing (MCC) for collaborative execution for executing compute-intensive workflows [Cuervo et al. 2010; Chun et al. 2011; Kosta et al. 2012; Gordon et al. 2012]. The MCC paradigm has attracted considerable attention both in academia and industrial community in recent times.

Several challenges remain to engage a mobile device as part of a computing infrastructure [Phan et al. 2002]. Some of these challenges are bandwidth, energy constraints, memory capacity, intermittent availability, proper incentive schemes against utilization, security, privacy, etc. In a controlled environment, some of these constraints can be addressed adequately in order to utilize the computation capacity of the mobile devices. For example, many of the reputed commercial organizations distribute smart phones among their senior employees [Agarwal 2011]. In such a corporate environment, it is possible to make it a policy that such phones be used for computation for the benefit of the company's infrastructure. Such a device can be used by the infrastructure whenever the device is present in the premises of the organization and is connected to the internal communication network. In such a scenario, the issues of communication reliability and cost, security, privacy are mitigated. To encourage such an environment, the organization may as well provide incentives in suitable forms. In the company of the authors, points are awarded for additional participation in company tasks (apart from regular assigned duties) and these points can be redeemed against purchases promoted by the organization.

In this paper, we adopt a simple localized mobile grid setting where the devices are accessible through a WiFi connectivity, and examine the problem of computation scheduling and workload management for improved timing performance. We consider a private company infrastructure with a gateway device and a mobile grid, with the gateway device hosting and assimilating an information database on which some

computation needs to be executed. The gateway device needs to decide on a computation scheduling and selection mechanism to engage the mobile devices and utilize their donated computation cycles. The primary objective driving this selection is to be able to finish execution of the application in the earliest possible time.

The gateway device is enabled with a task offloader which is the controller of the task selection framework. When the offloader wants to execute a task (in the form of a downloadable application), it invites bids from the owners of all devices connected to the offloader. Additionally, the offloader announces a deadline by which the computation has to finish. Associated with the task is a suitable reward to be earned by the selected bidder and a penalty. Each owner, intending to participate in the bid, executes a pre-installed analysis agent on his device. The agent takes as input the advertised application and the deadline and comes back to the owner with an advice whether to bid or not on the basis of its estimation of the execution time. In this paper, we consider the estimation of execution time of a task with various levels of information available about the device usage pattern. In our architecture, the mobile devices are active agents, who learn and build models of their owner usage patterns. The owner places a bid only if the estimated execution time is less than the advertised deadline of the job. The bid is the promised completion time within which the corresponding device can complete the advertised task. The offloader can possibly select one of the bidding agents for offloading the task based on some criterion. In the simplest case, it may choose the one with earliest promised completion time and offload the task to the selected device. We assume that the owners are rational (aware of penalty) and honest (no false bids). The interesting activity from the device's perspective is to analyze how/when/what to bid for, while designing the selection and scheduling mechanism is the offloader's challenge.

This paper is organized as follows. We present a motivating example in Section 2. Section 3 presents a model of the bidding and selection process. Section 4 describes our experiments while the following section presents related work. Finally, Section 6 concludes the paper.

2. MOTIVATION FOR THIS WORK

In this section, we present an example to illustrate the need for modelling a mobile device for its usage. A mobile device has various operational modes in its usage cycle. For example, when a user attends to a call, its communication modules are busy, when he listens to music or radio, its audio system is busy; when he watches a movie, its GPU remains busy. Manufacturers of mobile devices usually specify an operating model of their devices. An operating model is a state transition system where the states represent some high level operation modes (*e.g.* charging, audio on, network on, etc.) with average / maximum / minimum resource usage estimates when the device operates in that particular state, and possible interstate transitions. The operating condition of the device in these states can be attributed to its usage of processor, memory, cache, priority of the running jobs, battery power state, etc. The transitions in such a system are triggered by user interaction of the device and usage of device resources by various applications running in the system. In this paper, we extend this model to an usage-induced operating state model, a transition system based on the operating model and additionally, specialized by the usage pattern of the device owner.

In the context of exploiting a mobile device in our setting, we are interested in the availability of different resources in the device to utilize it for running an external computation. For simplicity, we assume here the states in the usage model of a device are characterized only by the percentage of CPU available. We also assume that the execution time of the advertised application on a given device architecture is known a-priori (or can be computed by dynamic simulation against a given data set or by using established methods like Worst Case Execution Time estimation [Wilhelm et al. 2008]). Such an estimate typically assumes full (100%) utilization of the resources in the device. In this work, we assume the execution time of the task is solely and linearly dependent on available CPU cycles in the devices. This essentially implies that if the execution time of a task is estimated as 10 time units, then the task is estimated to be complete in 20 time units when there are competing processes such that the job can avail only 50% of the CPU.

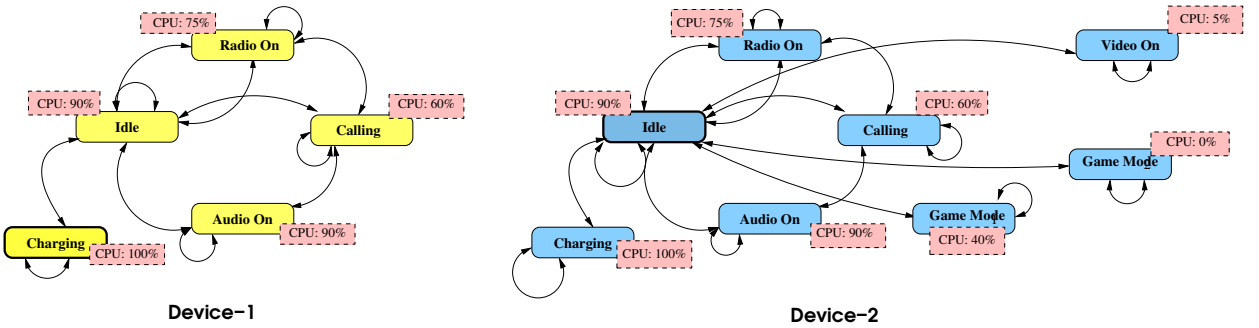


Fig. 1. Usage Model with CPU availability

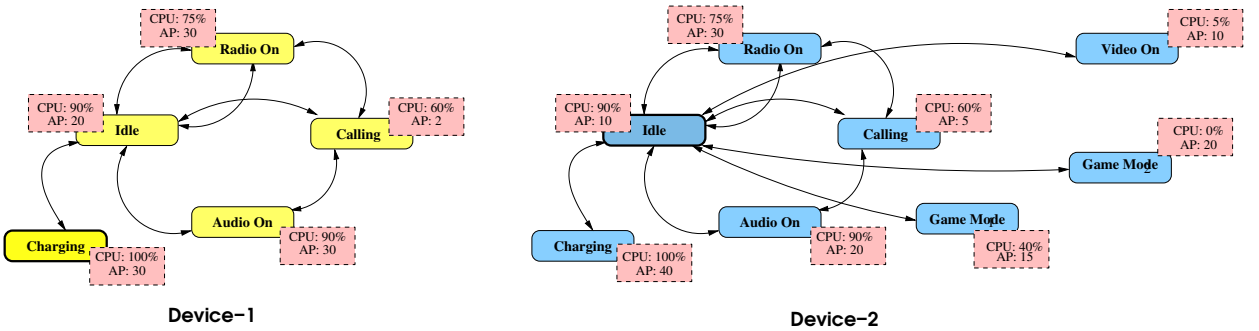


Fig. 2. Usage Model with Average Permanence

Table I. Execution on Device-1

State	CPU availability	Time in the State	Effective Execution
charging	100%	1 sec	1 sec
idle	90%	10 sec	9 sec
calling	60%	50 sec	30 sec
Completion Time :		61 sec	

As an example, we consider here a simple case of two mobile devices and one task to be offloaded to one of the devices. The task has a deadline of 60 time units. Each mobile device needs to estimate its bid based on its operational state model, as depicted in Figure 1. The events triggering the transitions are not shown in the figure, since they are not required for presentation of these examples. Each state in the state model is annotated with the fraction of CPU available for external computation at that state, which can be used for executing the external task. For the sake of simplicity, we assume here that the device takes one of the out-bound transitions from its current state, including the self loop, after every unit time. In other words, the device stays at each state for one unit of time, executes one of the outgoing transitions from the present state and moves to the next state (may be same as the current one) where it stays for one more unit, and this continues. We assume such transitions are instantaneous.

The Simplest Case. Both the devices have an estimate of the execution time of the advertised application on their architecture. Let us assume both of them come up with a value of 40 time units. When bids are invited, Device-1 is in the *charging* state and Device-2 is *idle*. If the devices always remain in the same state, the completion time of the task on Device-1 is 40 time units (100% CPU availability in *charging* state), while

Table II. Estimated Completion Time with best transition

Device-1				**Device-2**			
State	CPU availability	Time in the State	Effective Execution	State	CPU availability	Time in the State	Effective Execution
charging	100%	40 sec	40 sec	*idle*	90%	10 sec	9 sec
				charging	100%	31 sec	31 sec
Completion Time :		40 sec		Completion Time :		41 sec	

Table III. Estimated Completion Time with AP

Device-1				**Device-2**			
State	CPU availability	Time in the State	Effective Execution	State	CPU availability	Time in the State	Effective Execution
charging	100%	30 sec	30 sec	*idle*	90%	10 sec	9 sec
idle	90%	10.1 sec	10 sec	*charging*	100%	31 sec	31 sec
Completion Time :		41.1 sec		Completion Time :		41 sec	

that for Device-2 is $40 \times \frac{100}{90} = 44.44$ time unit. Thus Device-1 bids with a value of 40 and Device-2 bids with 44.44. Assuming the *offloader* awards the job to the one with earlier completion time, Device-1 is selected.

A more realistic scenario. In a more realistic setting, each device is expected to transit away from its current state during the job execution and therefore cannot guarantee constant CPU availability. In such a setting, the device can explore all possible paths in its state graph and optimistically choose a path which provides the best estimated completion time. Such a path obviously would go through states with high CPU availabilities. For example, Device-1 would consider the path involving only the *charging* state, which always guarantees it 100% CPU availability for the external job and can bid with value 40. On the other hand, Device-2 would consider the path from *idle* to the highest CPU available state, *i.e. charging*. Table II shows the estimated completion times in this case and the offloader may again select Device-1 for offloading.

Typically a state transition is triggered by external events, for example, incoming call, user's operation, etc. The execution paths chosen for bid as depicted above is therefore too optimistic. Consider the following scenario. At the time of execution of the external task, Device-1 remains in *charging state* for 1 time unit, in *idle* state for 10 time units and then moves to the *calling state* and remains there. The execution completion time is shown in Table I. The device thus completes 40 time units of computation in an effective duration of 61 time units and exceeds the deadline. This shows that only the best timing is not always a good candidate to decide on the bid, since a penalty is involved. A rational owner should ideally take this into account. On the other hand, a pessimistic strategy considering a maximal timing path may yield a completion time beyond the deadline. In either of the strategies, the path chosen for computation of a bid may not be the actual path taken during execution of the external task.

A more realistic estimate can be obtained by considering paths induced by the average usage by the user. To incorporate this, we further associate with each state an *average permanence* (AP) value [Li et al. 2011] and a transition probability on each out-going edge. AP implies the average time the device stays at the associated state. The revised model of the devices is depicted in Figure 2. Now we apply the same optimistic bid selection method based on the AP on states, assuming all transitions are equally likely. Also for each path we compute the probability of taking the path. This probability is a measure of confidence of the device taking that path. Device-1 chooses the path $charging \rightarrow idle$ which is associated with confidence value of 1 (since there is a single transition from *charging* state). The best confidence value ($1/8 = 0.125$ considering each of the 8 outgoing transitions are equally likely) for Device-2 occurs for the path $idle \rightarrow charging$. The completion time is computed in Table III. The offloader may choose Device-2 based on the better bid proposed by it. The example above assumes the transitions are equally likely. However in reality, they may

not be so, as we show in our experiments. We can learn the transition likelihood probabilities from user usage data and utilize them to enrich the bid above with these values.

3. PROPOSED METHODOLOGY

In this section, we present a detailed description of our approach. As discussed earlier, we have an application J with a deadline Δ. Each device has an usage based model as described below.

3.1 Usage Model of Device

We model the mobile device as a Probabilistic Finite State Machine with Average Permanence (PFSM-AP). The PFSM-AP model is defined as a tuple $\mathcal{U} =< \mathbf{S}, \mathcal{I}, \mathbf{T}, \lambda, \mathbf{H}, \mathbf{C} >$ where, $\mathbf{S}$ denotes the set of states; $\mathcal{I}$ is the set of external events; $\mathbf{T}$ denotes the transition function $\mathbf{T} \subseteq \mathbf{S} \times \mathbf{S} \times \mathcal{I}$; λ is the transition probability function, defined as $\lambda(s_i, s_j) = p_{ij}$where, $s_i, s_j \in \mathbf{T}$ such that, for each s_i the sum of the transition probabilities on its outgoing edges is 1; $\mathbf{H} : \mathbf{S} \rightarrow \Re$ is the average permanence function, defined as, $\mathbf{H}(s_i) = t_i$ where, $s_i \in \mathbf{S}, t_i \in \Re$ ($\Re$ is the set of reals); and $\mathbf{C} : \mathbf{S} \rightarrow [0, 1]$ is the CPU availability fraction (or equivalently availability percentage as used in the example).

ALGORITHM 1: Bid computation on a mobile device

```
input  : J : The task to be executed along with dataset
input  : Δ : Deadline for the tasks
input  : s : Present state of the device D_i
begin
1   Compute w_i of J;
2   if w_i > Δ then No bid and return ;
3   b_t ← ∅ // best time
4   Π_i ← paths (π_j^i) on U_i satisfying C1 and C2;
5   for each path p_j^i ∈ Π_i do
        if ρ^i(p_j^i) < ϒ then continue ;
        t ← δ^i(p_j^i);
        if (b_t > t) then b_t ← t;
6   Bid with b_t;
```

3.2 Detailed Methodology

The objective of the PFSM-AP described above is to characterize the device based on its free CPU cycles and duration the device is likely to remain in that state, as depicted in the motivating examples in Section 2.

Given a mobile device D_i with a PFSM-AP model $\mathcal{U}_i =< \mathbf{S}_i, \mathcal{I}_i, \mathbf{T}_i, \lambda_i, \mathbf{H}_i, \mathbf{C}_i >$, a task J with its dataset, the device needs to calculate its bid which can be presented to the offloader by the owner. Let us assume the task needs an estimated execution time of w_i on this device. As discussed earlier, this estimate is agnostic to the state model and assumes 100% CPU utilization. This is where PFSM-AP provides a better estimate. If $w_i > \Delta$, there is no point for the device to participate in the bid (intuitively there is no path in which the task can be completed within deadline even with 100% utilization all-through). The case is interesting only when $w_i \leq \Delta$. The principle behind our bid computation algorithm is as follows.

(1) Examine all possible paths in the PFSM-AP graph from the state the device is in, at the time when bids are invited

(2) Compute expected completion times on each of these paths considering that states in the path have different CPU availability
(3) Exclude paths where the expected completion time is greater than the deadline
(4) Exclude paths where the corresponding confidence value is less than some pre-determined threshold
(5) Determine a path which meets the deadline best and present the expected completion time on that path as the bid.

Execution Path Enumeration. Given the state machine of a device and the current state, there are potentially infinite number of paths from the start state. However, we are interested only in those paths where computation of the task can be completed within the advertised deadline. Since deadline is finite, such paths (excluding cycles involving states which offer 0% computing capacity) are also finite in number. Let us denote this set of paths as $\Pi_i = \{\pi_1^i, \pi_2^i, \ldots \pi_k^i\}$ for the device D_i where k denotes the number of such deadline-constrained paths. A path π_j^i is a state transition sequence, $< s_1^i, s_2^i, \ldots s_m^i >$, in the underlying PFSM-AP of D_i. We assume transitions to be 0-delay.

For each path $\pi_j^i =< s_1^i, s_2^i, \ldots s_m^i >$, we compute the following attributes which are useful for our algorithm.

—*Path execution time* ($\delta^i(\pi_j^i)$): The execution time of the application on the path π_j^i

$$\delta^i(\pi_j^i) = Z + \frac{w_i - Z}{\mathbf{C}(s_m^i)} \text{ where, } Z = \sum_{l=1}^{m-1} \left(\mathbf{H}(s_l^i) \times \mathbf{C}(s_l^i)\right)$$

The value of Z denotes the execution time on the first $m-1$ states on the path. The other term in $\delta^i(\pi_j^i)$ is the time required to finish the remaining fraction of work in the last state (s_m^i).

—*Confidence Value* ($\rho^i(\pi_j^i)$): The confidence value on the path π_j^i is computed as a product of the likelihood values on the transitions (assuming transition probabilities to be independent for simplicity) as below:

$$\rho^i(\pi_j^i) = \prod_{l=2}^{m} \lambda_i(s_{l-1}, s_l)$$

The following constraints are to be applied on valid paths to bound the search.

—*Task completion constraint*:

$$\mathbf{C1}: \quad \sum_{l=1}^{m} \mathbf{C}(s_l^i) \times \mathbf{H}(s_l) \geq w_i$$

where the term $\mathbf{C}(s_l^i) \times \mathbf{H}(s_l)$ denotes the quantum of computation done at the state s_l^i considering the average permanence and the CPU availability. The summation on the left hand side yields the total computation time on a given path. So the above constraint essentially limits our computation to paths whose time is more than w_i.

—*Deadline constraints*: Paths where the completion time of the task is more than Δ are not useful for bidding. Therefore,

$$\mathbf{C2}: \ \delta^i(\pi_j^i) < \Delta$$

We modify the standard depth-first traversal [Cormen et al. 2001] algorithm with constraints **C1** and **C2**, and also ignoring self loops involving a state with 0% CPU availability. These conditions bounds the length of the paths (step 4 of Algorithm 1) to finite value since w_i is finite. Therefore the algorithm terminates in

Table IV. Configuration of Mobile devices used in our experiment

		OS	CPU	
Device Name	**Model**	Android Ver	core @ clock speed	**Memory**
Samsung Galaxy	GT-S6802	2.3.6	Single Core @ 832 MHz	512 MB
Sony Experia L	C2104	4.1.2	Dual Core @ 1 GHz	1 GB
Micromax Canvas 2+	A110Q	4.2 (Jelly Bean)	Quad Core @ 1.2 GHz	1 GB
Google Nexus	Nexus-4	4.2 (Jelly Bean)	Quad Core @ 1.5 GHz	2 GB

finite time. The paths enumerated in the state graph are associated with different confidence values. A path with low confidence of traversing should be excluded to avoid penalties. An example of such a computation was presented in Sec 2. Algorithm 1 uses a threshold Υ to filter out such paths.

4. EXPERIMENTS AND RESULTS

Experiments with our proposed job-offloading technique were carried out in three phases on a set of seven mobile devices which seven of our employees volunteered to donate. The description of the devices are shown in Table IV. Our system had access to two Sony Experia devices, three Samsung Galaxy devices, and one each of Google Nexus and Micromax Canvas devices.

ALGORITHM 2: PFSM-AP Model Determination

begin
  `// Initial Clustering : Empirical Analysis`
1 **for** $i \leftarrow a$ **to** b **do**
    cluster data points in buckets of size i;
    $\delta_i \leftarrow$ deviation of cluster size values;
2 Choose i^* *s.t.* δ_{i^*} is the highest in $\{\delta_i : a \leq i \leq b\}$;
3 $\mathcal{C} \leftarrow$ cluster data points in clusters of size i^*;
  `// Reclustering`
  while *No change in cluster composition* **do**
4   $\{CC_k\} \leftarrow$ Compute cluster centers of $\mathcal{C}$;
5   Recluster data points around cluster centers $\{CC_k\}$ based on the distance of a point from cluster centers;
6   Remove a cluster if the size of the cluster is less than $\frac{\text{No of data points}}{|\mathcal{C}|} \times \frac{\Delta}{100}$;
7 Each cluster is a state and the mean value is the percentage of the free CPU cycles;
  `// Transition and Transition Probabilities`
8 Traverse the data and compute average time in a state;
9 Traverse the data and compute number of transitions for all pairs of states;
10 Compute the transition probability of an edge $s \rightarrow d$ as the fraction of transitions from state s to d against all transitions out of state s, *i.e.* , $\frac{\text{No. of transitions from } s \text{ to } d}{\sum_{\forall p} \text{No. of transitions from } s \text{ to } p}$

4.1 Model Generation

During the first phase of the experiment, we worked towards building the PFSM-AP models of the mobile devices participating in our experiments. We developed and installed a small android application which collects device usage trace data (like free memory and CPU usage) for every second and logs into the devices. The users carried this application, active in their devices, and the application gathered data for a week. We then collected this data and analyzed them offline to discover PFSM-AP models. We extracted the percentage of free CPU cycles only from the data and applied clustering to build the PFSM-AP. The outline of the clustering technique is shown in Algorithm 2.

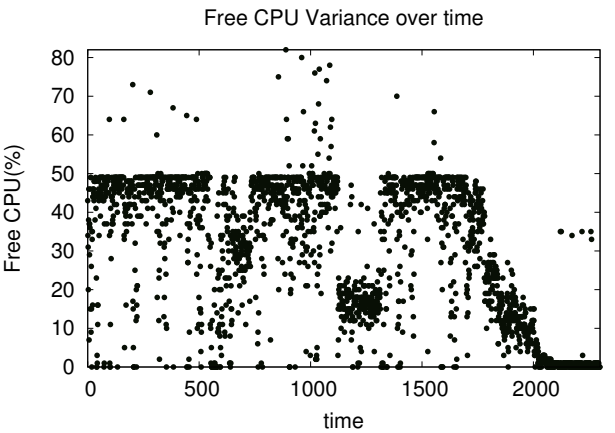


Fig. 3. CPU Usage Pattern

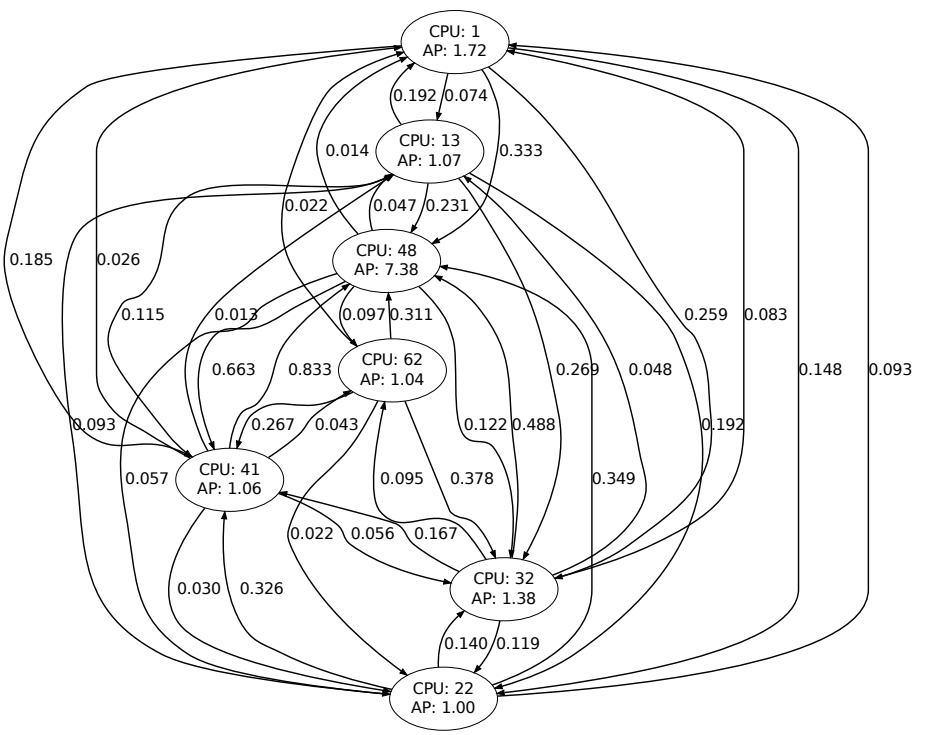
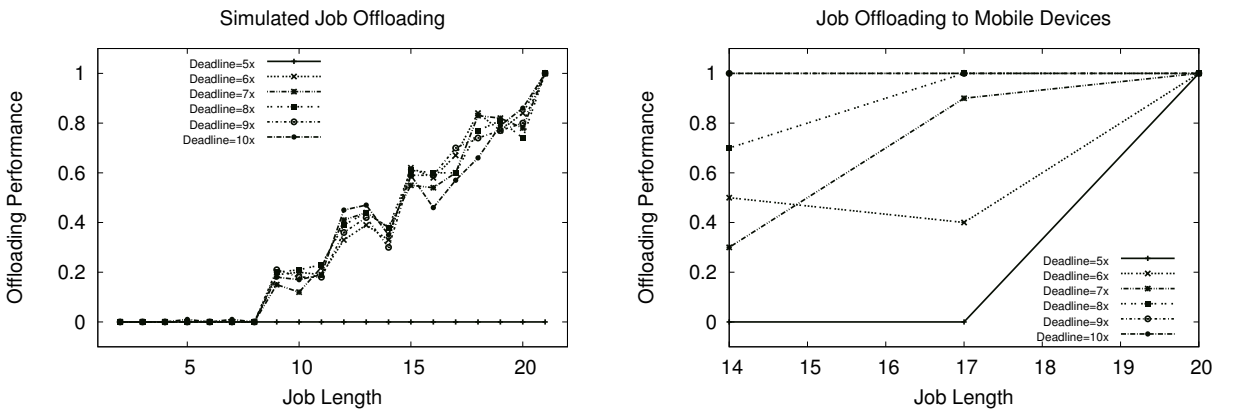


Fig. 4. PFSM-AP Model

The first phase of the algorithm creates an initial set of clusters based on the CPU availability (data gathered as % of free CPU cycles). In this part, we create buckets of different sizes and assign data points to these buckets. For example, when bucket size is 2, data points in $[0, 1]$ are put into one bucket, points in $[2, 3]$

are put in another bucket, and so on. Since the CPU availability values are in $[0, 100]$, we will have 50 buckets to be considered in this case. After creation of these buckets, we compute the deviation of the number of points in these buckets. The exact values of a and b are chosen depending on the number of observations. It is intuitively obvious that a very high bucket size will create too few clusters. Then we choose the bucket size where the deviation is the highest. Empirically this captures points which are close in one cluster. This serves as an initial cluster to be refined in the subsequent phase of the algorithm. In the next phase, we identify clusters (formed in the initial stage) where membership count is low. The parameter Δ is used as the threshold of count of data points in a cluster and any cluster having number of data points less than the threshold are removed and data points in this cluster are reassigned to other clusters. The threshold is Δ percent of the average data points in clusters. The Δ value is small and in our experiment $\Delta = 5$. The value indicates cluster removal threshold is 5% of the average cluster size. We applied this clustering algorithm on the system traces collected from mobile devices by our mobile application and constructed the PFSM-AP models of these devices. Figure 3 shows a part of the CPU usage pattern of one of the users and the PFSM-AP model constructed thereafter is shown in Figure 4.



(a) Simulated System Performance

(b) Performance on real devices

Fig. 5. Experiment Results

4.2 Simulation of Offloading System

We developed a simulation system to observe the behavior of our task offloading infrastructure. We simulated the offloader system and also the task execution on device VMs. The VMs simulate the PFSM-AP models generated from the traces on the mobile devices. The simulated offloader generated tasks of various kinds to be offloaded to these devices. When awarded a task, a device simulates task execution while simulating the state transition based on its model. For each task type, 100 similar tasks were generated and offloaded to devices. The number of these tasks successfully completed on the devices (*i.e.* completed within the given deadline) are recorded and used for computing performance of the offloading method. The performance of the system is simply the fraction of the offloaded jobs successfully completed by the bidding device.

In our simulation system, the offloader generates jobs of various durations, assigns various deadlines to these jobs, invites bid and submits the job to the winning device. Figure 5(a) shows the job offloading performance for job execution times ranging from 4 to 29, and deadlines varying from 2× to 10× of the job execution time. The horizontal axis of the figure represents variation in job execution duration, while the vertical axis represents the offloading performance. The offloading performance, as discussed earlier, is

the fraction of the number of jobs the infrastructure could complete by offloading them to winning devices and the devices subsequently could complete execution within given deadline. A value of 0 as performance indicates that the infrastructure was unable to effectively utilize any device for computation. On the other hand a value of 1 indicates the infrastructure could execute all jobs using the devices. Please note that in our experiment the infrastructure offloaded one job at a time and concurrent offloading was not considered.

It is evident from the experiment that, when task runtime is low, success rate of task-offloading is low as well. Also when the deadline is very tight in comparison to execution time of the task, task offloading is not beneficial. When deadline is tight, if the device cannot operate with near 100% CPU availability all the time, the corresponding completion time is more likely to overshoot the task deadline. When the deadline is very relaxed (*e.g.* approximately 7× that of job execution length) offloading works well and is beneficial.

4.3 Working with real devices

In this phase, we evaluated the performance of our offloading system with an offloader which has the seven devices, described earlier, for it to exploit. For this experiment, we used an application which estimates the value of π, which is written as a native android application. The application is a compute intensive one. Longer the application runs, the estimation is better. We conducted the experiment for various job durations and the job duration was varied by changing the desired accuracy of π calculation. Figure 5(b) shows the job offloading performance of the system. The result of this experiment shows that jobs with relaxed deadlines are good candidates for offloading.

5. RELATED WORK

In recent times, there has been significant research on the theme of Mobile Cloud Computing (MCC) which propose the use of a collaborative computing infrastructure consisting of mobile devices and a backend cloud. MAUI [Cuervo et al. 2010] and CloneCloud [Chun et al. 2011] are two notable systems which use a backend computing infrastructure for collaborative job execution. Several other articles as well address the problem of workflow partitioning in an MCC setting [Rahimi et al. 2013]. The basic intuition behind these partitioning strategies is to decide on the best platform (mobile device or cloud) to execute each sub-task of a given workflow with an objective of optimizing the cost (in terms of energy, time, communication, etc).

Authors in [Park et al. 2003] address the problem of intermittent disconnection and analyzes the problem using Markov-chain model. Markov Decision Process (MDP) has been used to model the behavior of mobile devices to achieve objectives like optimization of power usage [Jung et al. 2010]. However in our case, we resort to a simpler model since we do not need the full capabilities of an MDP for this problem. A comprehensive survey of Mobile Cloud Computing (MCC) can be found in [Dinh et al. 2011]. Systems like *Misco* [Dou et al. 2010] and *Hyrax* [Marinelli 2009] extend MapReduce so that computation capabilities of mobile devices can be utilized. *Serendipity* is a task dissemination system over a mobile grid [Shi et al. 2012]. The system relies on collaboration among mobile devices using WiFi connection to share collective computation power. The design of the system accepts that disconnection of devices is a norm and its underlying architecture incorporates the assumption. We consider a more generic usage model driven scenario in this work.

6. CONCLUSION AND FUTURE WORK

Proposals of using mobile devices to augment computing infrastructure have been proposed in literature. In this paper we present a basic scheme of job offloading on a mobile grid. Mechanisms for automatic learning of likelihood probabilities, using more advanced models for analysis (*e.g.* MDP), execution time estimation, designing more effective bidding, reward-penalty schemes may be looked into for future explorations. In this paper we also assume the network and the devices are reliable. Issues of fault tolerance in this context are our future research agenda.

REFERENCES

Ankit Agarwal. 2011. Enterprise Smartphone Usage Trends. http://bit.ly/loIqE1. (June 2011).

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.

Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*. ACM, 301–314.

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. *Introduction to Algorithms* (2nd ed.). McGraw-Hill Higher Education.

Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 49–62.

Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. 2011. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing* (2011).

Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H Tuulos. 2010. Misco: a MapReduce framework for mobile systems. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*. ACM, 32.

Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. 2012. Comet: code offload by migrating execution transparently. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, OSDI*, Vol. 12. 93–106.

Eric Jung, Frank Maker, Tang Lung Cheung, Xin Liu, and Venkatesh Akella. 2010. Markov decision process (MDP) framework for software power optimization using call profiles on mobile phones. *Design Automation for Embedded Systems* 14, 2 (2010), 131–159.

Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. 2012. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 945–953.

Xiaoyue Li, Alison Gray, Daqing Jiang, and Xuerong Mao. 2011. Sufficient and necessary conditions of stochastic permanence and extinction for stochastic logistic populations under regime switching. *J. Math. Anal. Appl.* 376, 1 (2011), 11–28.

Eugene E. Marinelli. 2009. Hyrax: Cloud Computing on Mobile Devices using MapReduce. (Sept. 2009).

Arijit Mukherjee, Himadri Sekhar Paul, Swarnava Dey, and Ansuman Banerjee. 2014. ANGELS for distributed analytics in IoT. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 565–570.

Sang-Min Park, Young-Bae Ko, and Jai-Hoon Kim. 2003. Disconnected operation service in mobile grid computing. In *Service-Oriented Computing-ICSOC 2003*. Springer, 499–513.

Thomas Phan, Lloyd Huang, and Chris Dulan. 2002. Challenge: Integrating Mobile Wireless Devices Into the Computational Grid *(MOBICOM-2002)*. 271–278.

M. Reza Rahimi, Nalini Venkatasubramanian, and Athanasios V. Vasilakos. 2013. MuSIC: Mobility-Aware Optimal Service Allocation in Mobile Cloud Computing. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD '13)*. IEEE Computer Society, Washington, DC, USA, 75–82.

Sharif Sakr. 2011. NVIDIA says Tegra-3 is a "PC-class CPU". http://engt.co/srvibU. (2011).

Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. 2012. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM, 145–154.

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, and others. 2008. The worst-case execution-time problem - Overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)* 7, 3 (2008), 36.