Chapter 4
A Framework for Speculative Job Scheduling on Mobile Cloud Resources

Ansuman Banerjee, Himadri Sekhar Paul, Arijit Mukherjee, Swarnava Dey, and Pubali Datta

4.1 Introduction

Recent advances in mobile technology have enabled immense penetration of these devices in the common market. At the same time, ancillary businesses revolving around mobile devices have attained a boost of similar magnitude. With all these developments both in the technical front and in business, mobile devices are poised to revolutionize the personal computing landscape. The mobile application market alone is estimated to reach US$77 billion by the end of 2017 [1]. Although computing capacity of mobile devices has significantly increased over the past decade [2], so did the computation demand from their users. The complexity and therefore the computation requirement of the mobile applications have increased in similar pace over time to keep up with user expectations. A typical smartphone performs various tasks, including management and responses to user interactions. Typically there are several computation hungry tasks which run in the background to enhance user experience with its device. Since a mobile device is a very personal device, user's experience with its device is of prime importance.

This work was done during Pubali's association with TRDDC, Pune, India

A. Banerjee
ACMU, Indian Statistical Institute, Kolkata, India
e-mail: ansuman@isical.ac.in

H. S. Paul (✉) · A. Mukherjee · S. Dey
TCS Research and Innovation, Kolkata, India
e-mail: himadriSekhar.Paul@tcs.com; mukherjee.Arijit@tcs.com; swarnava.dey@tcs.com

P. Datta
Department of Computer Science, University of Illinois Urbana Champaign, Champaign, IL, USA
e-mail: pdatta2@illinois.edu

S. Hu, B. Yu (eds.), *Big Data Analytics for Cyber-Physical Systems*,
https://doi.org/10.1007/978-3-030-43494-6_4

Choice of applications running on mobile devices is based on user preferences. The pattern of usage of a device is also specific to a user. Researchers have attempted to discover user specific patterns in usage of a device. Study of such patterns is important in various contexts, including network usage [3], battery charge decay characterization [4], etc. Shye et al. introduced a Markov decision process-based model to capture user activity [5]. In this paper, we model the usage pattern of a device as a state transition system. Our model is simple, yet effective, in the contexts for which it was used in our experiments. We used this model in scheduling background tasks in a mobile device and also in the context of collaborative computing in mobile cloud.

Similar to a regular desktop computing environment, a smartphone also performs several routine jobs in the background to keep its computing environment up-to-date and enhance user experience with its device. A smartphone performs system updates in regular intervals, runs virus scans at regular intervals, builds file indexes, mines logs (like call logs) to build knowledge bases, etc. There may be several such background activities, which are important and yet low-priority jobs for the scheduler, which help in creating a comfortable computing environment for its user. Most of these background activities are resource hungry and can consume considerable amount of CPU cycles while running. It is not an uncommon experience that such jobs trigger off in uncanny hours when the user is very active with its device, resulting in delayed response from the application the user is using. There are usually trapdoors available for a user to specify a schedule for these activities individually. Such schedules are static and do not cater to the dynamic nature of usage of the device. In this paper, we propose to employ the usage model to intelligently suggest when to run a background task, such that demand for resources is evenly distributed temporally, resulting in better user experience with the device.

In this paper, a state transition model of user usage pattern is used as a guidance for estimating execution time of a given task. The model is also used to determine a schedule of the task in question, such that sufficient resource is available to the background task without overloading the system and also to ensure that the user experience with the device is not affected. This basic technique can be applied in a different form, in the context of collaborative computing involving mobile devices. With global penetration of mobile devices in the commercial market, the count and capacity of the devices are in the rise. To harness the free computation cycles of these devices, several collaborative computing frameworks have been developed, namely Hyrax [6], Misco [7], Serendipity [8], etc. In this paper, we adopt a localized mobile grid setting where the devices are accessible through a WiFi connection. We examine the problem of computation scheduling and workload management for improving timing/energy performance. We consider a private company infrastructure with a gateway device and a mobile grid, where the gateway device is expected to host and assimilate an information database on which some computation need to be executed. The gateway device needs to decide on a schedule of computation and a selection mechanism so as to engage the mobile devices and utilize their donated computation cycles. The primary objective of driving this selection is to be able to finish execution of the application at the earliest possible time.

The paper is organized as follows. In Sect. 4.2, we present a model of pattern of usage of a mobile device and also present a discussion on how to extract such a model from real usage traces. Section 4.3 presents a scheduler, augmented by the usage model, such that the usage experience with the device is not affected by execution of background jobs and present some experiment results in Sect. 4.4. We use the usage model in a mobile and cloud collaborative computing setting and present motivation for the same in Sect. 4.5. We present such a collaborative model based on bidding in Sect. 4.6 and our experiments in Sect. 4.7. Finally, Sect. 4.9 concludes this paper.

4.2 Modelling Usage Patterns

A mobile device is probably the most personal device that its owner carries and the device is most personalized by its owner. The operation of such a device carries its owner's signature. It has been observed that there is a pattern of use for each device owner and based on this assumption, researchers have tried to model the usage pattern of a device on various aspects. One of the objectives of such a model is to predict the temporal variation of certain aspects of the device. In this paper, our objective is to estimate the execution time of a given task utilizing free computation cycles of the 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 $\mathscr{U} =< \mathbf{S}, \mathscr{I}, \mathbf{T}, \lambda, \mathbf{H}, \mathbf{C} >$ where,

- $\mathbf{S}$ denotes the set of states.
- $\mathscr{I}$ is the set of external events.
- $\mathbf{T}$ denotes the transition function $\mathbf{T} \subseteq \mathbf{S} \times \mathbf{S} \times \mathscr{I}$.
- λ is the transition probability function, defined as

$$\lambda\left(s_i, s_j\right) = p_{ij} \quad \text{where, } s_i, s_j \in \mathbf{T}$$

  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,

$$\mathbf{H}\left(s_i\right) = t_i \quad \text{where, } s_i \in \mathbf{S}, t_i \in \Re$$

  $\Re$ is the set of reals.

- $\mathbf{C} : \mathbf{S} \rightarrow [0, 1]$ is the CPU availability fraction or equivalently availability percentage.

The objective of the model depicted as above is to characterize a device based on its free CPU cycles. However, the model is not restricted to this feature only and can easily be extended to include any other feature of interest. We restrict our model to free CPU cycles since we use only this feature in our estimation of execution time, as described below.

4.2.1 Battery Decay

The source of energy of a mobile device is of primary consideration and the rate of decay of charge plays an important role in its schedule. For example, if an application takes such a long time to execute in the mobile that it is likely to exhaust all its energy, it is not advisable to schedule that task. The primary source for battery power consumption in a mobile phone has been identified as its screen and communication modules [9, 10]. There have been several interesting proposals of optimizing power consumption in mobile devices based on energy profiles of the communication devices and scheduling data volume transfers [11–13]. Authors in [14] analyze and present power consumption results based on usage pattern of mobile devices. The authors claim that primary sources of the power decay in mobile devices are active screen and CPU. Their study reveals that majority of the usage patterns have long intervals between two subsequent active screens. Power modelling is an involved field of study involving several sub-areas like architecture, operating system, software engineering, etc. Instruction level power models have been proposed in literature to estimate the power consumption by applications [15, 16]. The objective of these models is to accurately characterize an application for their power consumption which can be used for power optimization of the application. Authors in [4] propose a method for usage pattern-based estimation of battery power. They use an auto-regression model on logged usage patterns to predict device power usage. However, in our context, we only require a coarse-level estimation of power consumption by an application, without performing such expensive profiling techniques. In this work, our objective is to utilize free computation cycles of a mobile device and we assume that computation power of CPU does not change due to dynamic voltage and frequency scaling (DVFS) level. In this work, we use a simple model to characterize decay of battery charge in mobile devices. For the sake of simplicity of illustration, we assume here that the expenditure in battery energy by an application is linear in the number of CPU cycles it consumes.

$$\mathbf{B} = \beta \times n$$

where n is the number of CPU cycles required by an application, $\mathbf{B}$ is the energy required for n cycles, and β is a known constant

4.2.2 Generation of PFSM-AP Model

Generation of the usage model was the first phase of our experiment. As part of model generation, we chose a relatively small history of usage of a device to model recent usage patterns of the person. Usage is known to vary widely over time, since interests and need of the person change over time. During this phase, we captured usage patterns of seven mobile devices owned by seven of our employees who volunteered to donate their devices for our experiments. The description of the devices is shown in Table 4.1. These devices include two Sony Xperia devices, three Samsung Galaxy devices, and one each of Google Nexus and Micromax Canvas devices. During this 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 can collect device usage trace data (like free memory and CPU usage) every 5 s and log into the devices. The users carried this application, active in their devices, and the application gathered data for approximately a month. We then collected this trace data and analyzed them offline to build their corresponding PFSM-AP models. The trace data for the last day, however, was not considered for building the model, but was replayed during our experiments with this model. We extracted the percentage of free CPU cycles only from the data and applied clustering to build the PFSM-AP. The percentage of free CPU cycles of a state was calculated as the mean of the data points. Once the state transition model was built, we also derived, from the log, the durations the device remained in a certain state. The permanence value of the state was computed as the average of these durations. The method is outlined as Algorithm 1.

The clustering heuristic is a form of density-based clustering and works iteratively to refine a set of initial clusters. The heuristic has two phases. The first phase of the clustering heuristic creates an initial set of clusters based on the feature of the CPU availability value (data gathered as the percentage of free CPU cycles). In the initial clustering phase (line 1 of Algorithm 1), we create some initial clusters based on the range of CPU availability value. In this phase, we create a set of *set of clusters*, based on different bucket sizes. A bucket is defined on a range of values for a feature, in this case, the CPU availability value. For example, consider buckets of size 2. Data points with CPU availability value in the range of [0, 1] (i.e., the first bucket) are put into one cluster, points in the range of [2, 3] (i.e., second bucket) are put into a different cluster, and so on. Since the CPU availability values are in

Table 4.1 Configuration of mobile devices used in our experiment

Device name	Model	OS Android Ver	CPU Core @ clock speed	Memory
Samsung Galaxy	GT-S6802	2.3.6	Single Core @ 832 MHz	512 MB
Sony Xperia 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

Algorithm 1: PFSM-AP model determination

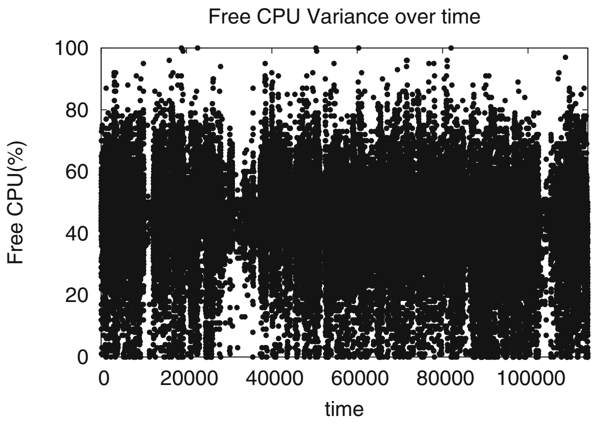
```
   begin
       // Phase 1: Initial Clustering - Empirical Analysis
1      for $i \leftarrow a$ to $b$ do
           cluster data points in buckets of size $i$;
           $\delta_i \leftarrow$ deviation of cluster size values;
2      Choose $i^*$ s.t. $\delta_{i^*}$ is the highest in $\{\delta_i : a \leq i \leq b\}$;
3      $\mathscr{C} \leftarrow$ cluster data points in clusters of size $i^*$;
4      $\{CC_k\} \leftarrow$ Compute cluster centers of $\mathscr{C}$;
5
       // Phase 2: Refinement and Reclustering
6      while No change in cluster composition do
7          Recluster data points around cluster centers $\{CC_k\}$ based on the distance of a point
           from cluster centers;
8          Remove a cluster if the size of the cluster is less than $\frac{\text{No of data points}}{|\mathscr{C}|} \times \frac{\sigma}{100}$;
9          Reassign data points of removed clusters to existing clusters based on the distance
           of a point from cluster centers;
10         $\{CC_k\} \leftarrow$ Compute cluster centers of $\mathscr{C}$;
11     Each cluster is a state and the mean value is the percentage of the free CPU cycles;
12
       // Phase 3: Computation of Transitions and Transition
          Probabilities and Creation of PFSM-AP Model
13     Traverse the data and compute average time in a state;
14     Traverse the data and compute number of transitions for all pairs of states;
15     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. $\frac{\text{No. of transitions from } s \text{ to } d}{\sum_{\forall p} \text{No. of transitions from } s \text{ to } p}$
```

the range [0, 100], we can construct 50 buckets and therefore 50 initial clusters can be constructed for a bucket size of 2. Then we compute the deviation of the CPU availability value for each cluster. Such sets of cluster-sets are created for all bucket sizes in the range of $[a, b]$. The exact values for 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. Out of these sets of cluster-sets, we choose the one where the deviation is the highest (line 2 of Algorithm 1). Empirically the chosen cluster-set captures densely packed points into a single clusters. Such a set serves as an initial set of clusters to be refined in the subsequent phases of the heuristic.

In the next phase, we refine the clusters and their memberships. The refinement is done by modification of clusters with low membership count. Any cluster having number of data points less than a threshold is removed. The threshold is defined as σ percent of average membership count of all the clusters. The value of σ was taken as a parameter to the algorithm. Typically the value of σ is small and in our experiment σ was chosen as 5 which indicates that clusters with less than 5% of the average cluster size are modified. Then a cluster refinement is done as follows. All the data points are reclustered based on their distances from the centers of the surviving clusters. Once reclustering is complete, the cluster centers are recomputed.

The last phase of the algorithm is the construction of the PFSM-AP model. The process is straightforward. Each of the clusters constructed in phase-2 represents a state in the model. The clusters are enumerated and the enumeration values are the cluster-ids or state numbers. Each of the entries of the mobile trace data now can be annotated with a state number. Since the trace data entries also contain time-stamps, it is easy to determine the duration the device stays in a certain state. For this analysis, the computation of average permanence for each state is also simple. Once the trace data is annotated with state numbers, it is also easy to define state transitions and assign a transition probability, as shown in line 15. Figure 4.1 shows a part of the CPU usage pattern of one of the users, and the PFSM-AP model constructed thereafter is shown in Fig. 4.2.

Fig. 4.1 CPU usage pattern



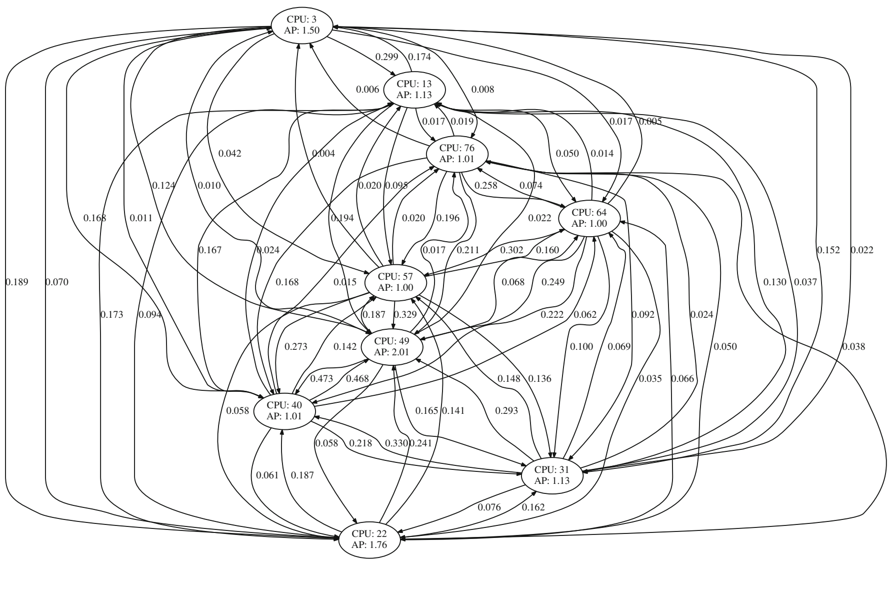


Fig. 4.2 PFSM-AP model

4.2.2.1 Analysis of the Heuristic

The heuristic presented as Algorithm 1 has two loops. The for loop at line 1 indexed by variable i is bounded by variables a and b. We now show that the while loop at line 6 executes finitely many times. The while loop is executed until no data point is reassigned to a different cluster (through cluster modification). Assignment of a data point to a cluster changes due to two effects: (1) One or more clusters are modified (line 8), and (2) the cluster center shifts, changing the membership of the data points (line 9). Let us consider the first case. For any given $i^* : a \leq i^* \leq b$, the number of initial clusters is bounded. Since the algorithm only allows small clusters to be destroyed and no new cluster is created inside the while loop, the cluster removal step (line 8) is bounded. Therefore, the number of cluster reassignments for a data point due to cluster modification is also bounded. Now, let us consider the second case. Cluster centers can shift when new data points join a cluster or data points are removed from the cluster. Clustering of data points around cluster centers is an optimization problem where minimization of distances of data points from their corresponding cluster centers is the objective function. This also executes for finitely many iterations.

4.2.3 Execution Estimation Model

We assume that the execution time of a given job on a given device architecture is known a priori (i.e., can be computed by dynamic simulation against a given dataset or by using established methods like worst case execution time estimation [17]). 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. In the following section, we apply this model to augment the device scheduler to intelligently schedule background tasks to improve user experience with the device.

4.3 Usage Model-Based Scheduler

Computation capacity of modern mobile devices is increasing and is poised to replace desktop computing devices [2]. Mobile devices run broadly two types of tasks, foreground jobs with which the user interacts and background jobs which are essentially maintenance tasks. The schedulers of these OSes are responsible for prioritizing foreground jobs so that users experience minimum delay. Maintenance tasks, like virus scans, system updates, building file index, etc. which are not part of typical usage of devices, run as low-priority jobs in the background, consume much less system resources, yet are important for the operation of the device. The usage model of a device, described in Sect. 4.2, captures its resource availability under regular usage by its owner and its variation over time. Many of the background tasks are periodic tasks and usually the OS provides means to the user to schedule them, such that they are triggered when the device is relatively free. Yet it is a common experience that such tasks are triggered at times when the user is in active use with the device. This results in a race for resource acquisition and results in slower response. To improve user experience in such cases, it is important to determine a schedule of such tasks when the usage of resources is likely to be low. A scheduler can leverage the usage model to intelligently schedule background tasks such that the user can more comfortably use its device without being affected/annoyed. In this section, we present a model of a scheduler which is aware of the usage pattern of the device.

4.3.1 Model of Scheduler

We consider a mobile device with usage pattern $\mathscr{U} =< \mathbf{S}, \mathscr{I}, \mathbf{T}, \lambda, \mathbf{H}, \mathbf{C} >$ as presented in Sect. 4.2. There is a set of background jobs to be scheduled by the scheduler, opportunistically. Each of these jobs have deadlines associated with them, and we want to maximize the number of jobs which successfully complete within their deadlines. Let $\mathbf{J} = \{J_1, J_2, \ldots J_n\}$ denote a set of background jobs. The deadline of each of the jobs is denoted as $\mathscr{D}(J_i)$. Also we denote by $\mathscr{E}(J_i)$ the estimated execution time of the task on the device.

The selection process of a background job from the set $\mathbf{J}$ for execution is based on the present computation state and battery state of the device. The objective of the process is to identify a job which will fit the device's energy budget and deadline of the job, without hindering the operations done by the user on the device. We present an outline of the steps for the selection process below.

1. Sort jobs in $\mathbf{J}$ in increasing order of their deadlines.
2. Choose a path, $P =< s_0, s_1, s_2, \ldots s_m >$ in PFSM-AP model, such that for every edge (s_{i-1}, s_i) in P, $\lambda(s_{i-1}, s_i)$ is highest among all the outgoing edges from s_{i-1}.
3. Selection of job: for all jobs P_i, taken sequentially from the sorted job list, do the following:

    • Projection of execution: Calculate the time required to execute J_i on path P such that,

        (a) only α-fraction of free CPU of every state in P is used for execution of J_i
        (b) execution of J_i on path P completes within $\mathscr{D}(J_i)$
        (c) decay of battery on this execution time does not result in complete drainage of the battery.

    Schedule J_i immediately if all the above conditions are met.

We preserve α-fraction of the free resources as buffer in each state of the PFSM-AP and this parameter can be used to control free resource usage by background jobs. Therefore, a job can only utilize α-fraction of the free computation cycles of the device so that the user's experience with its device is not degraded. Since a device cannot offer a steady computation power to a background job, since it ensures that user's jobs have higher access priority over CPU, the background job executed in a device experiences variable computation power. The classical version of this problem is equivalent to *open shop scheduling* with multiple jobs and single machine, and it is known to be NP-hard [18]. In contrast to the classical open shop scheduling problem, the computation power of the machine varies over time.

The PFSM-AP model presented here models the varying computation power available for a background job. Each state of the PFSM-AP model represents available resource in that state. Given an initial state in the PFSM-AP model which represents the present computation state of the device, the selection process needs to refine $\mathscr{E}(J_i)$, the estimated run-time of the job J_i on the device with no other computation load. We present an on-line heuristic as Algorithm 2, which examines the jobs, sorted in their ascending order of their deadlines (i.e., job with the nearest deadline as higher priority for scheduling) and estimates whether the job can be completed within its deadline before the battery is drained out. The estimation process is based on the estimation of execution time of J_i, which shares CPU with other jobs being initiated by the user all of which are treated as higher priority jobs than J_i.

For the process of refinement of execution time of a given job, one has to determine a path in the model, given an initial state. Refinement, in our case, is essentially a projection of $\mathscr{E}(J_i)$ on the path, based on the available computation cycles in the path and is computed in lines 4–6 of Algorithm 2. Therefore, the choice of the path determines the accuracy of the refined estimation of the execution time. The edges of the PFSM-AP model are annotated with probability of state transition by the CPU. For each of these paths, we are interested in the shortest path-prefix which can accommodate the computation requirement of the job. Since we are only interested in a path-prefix which the device is most likely to follow, we need to choose one which has the highest probability value, computed as the product of the constituent edges. Evidently the problem is computationally hard but we require to solve it efficiently to be able to quickly choose a job from the list. The heuristic presented here is based on the assumption that user behavior on its device is frequently repeated. So we always follow the outgoing edge having the highest probability value (line 7).

4.3.1.1 Analysis of the Heuristic

The heuristic has two loops nested. The outer loop at line 1 takes one job from a list and refines its execution time in the inner loop at line 3. The execution for the

Algorithm 2: $\alpha Sched_1$: Scheduling of a job with one deadline

```
  input : $\mathbf{J} = \{J_1, J_2, \ldots J_n\}$ : Set of jobs to be executed
  input : $\mathscr{U}$ : PFSM-AP model of the device
  input : $\mathbf{s}$ : Present state of the device
  input : $\mathscr{B}$ : Estimated battery charge level
  output: $X$ : Job to be scheduled
  begin
      $\mathbf{J}' \leftarrow$ Sort $\mathbf{J}$ in increasing order of their deadline i.e., $\mathscr{D}(J_i)$;
1     foreach $J_i \in \mathbf{J}'$ do
2         $T \leftarrow \mathscr{E}(J_i)$: estimated computation time of $J_i$;
          $tm \leftarrow 0$ // Wall clock : Initialization
3         repeat
4             $c \leftarrow (1-\alpha) \times \mathbf{C}(\mathbf{s}) \times \mathbf{H}(\mathbf{s})$;
5             $T \leftarrow T - c$;
6             $tm \leftarrow tm + \mathbf{H}(\mathbf{s})$;
7             $e \leftarrow (\mathbf{s}, r)$ where $\lambda(\mathbf{s}, r)$ is maximum among all outgoing edges from $\mathbf{s}$;
              $s \leftarrow r$;
              // Exceeds deadline - not feasible
              if $tm > \mathscr{D}(J_i)$ then continue at 1
              // Exhausts battery - not feasible
              if $\mathscr{B} \leq (\beta \times tm)$ then continue at 1
          until $T \leq 0$;
8         Schedule $J_i$ for execution and return;
```

outer loop is bounded by $|\mathbf{J}|$. The inner loop essentially traverses the PFSM-AP model from the initial node, given as input to the heuristic, following the edge with the highest probability value. The loop is terminated when the remaining execution time of the job, T, drops beyond 0. In each iteration, the value T monotonically decreases. The PFSM-AP model can possibly include at-most one state with 0% available computation capability, due to clustering process. Also by construction of the PFSM-AP model, no state can have a self-loop. A self-loop indicates that the device remains in the same state after a duration. However, the whole duration for which the device remains in a state is computed as permanence for that duration. By model construction, the average of all permanence values is computed and stored as *average permanence* for the state. Given a self-loop free graph of the PFSM-AP model, at-most one state (with 0% CPU availability) at which T value decreases is chosen and the value of T monotonically decreases in each iteration of the inner loop. Thus, the algorithm terminates in a finite number of steps. In the next section, we present comparison of our schedule with shortest job first (SJF) schedule [19] with priority in a simulated environment.

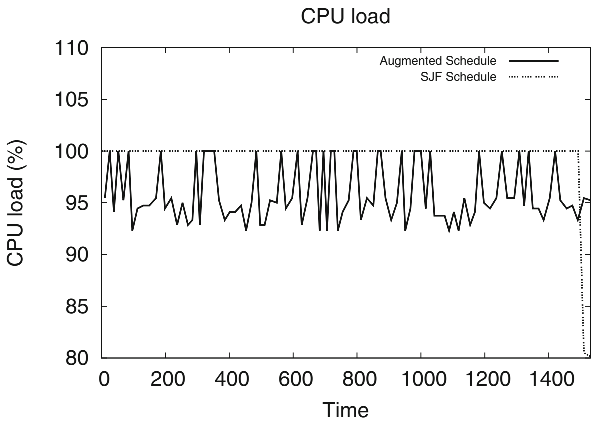


Fig. 4.3 Comparison of usage pattern augmented scheduler

4.4 Result: Usage Model-Based Scheduling

We carried out simulation experiment to compare our heuristic with the SJF schedule. Our technique utilizes the usage pattern captured during our first phase of experimentation described in Sect. 4.2.2. We used the PFSM-AP model to augment our scheduler. We simulated the mobile devices and implemented our heuristic as an augmentation of the shortest job first scheduler. The background jobs were simulated to be of varying duration from the range [10, 50] s with their deadlines to be twice as their execution time, and we simulated 100 such jobs of low-priority. We used $\alpha = 0.2$ as specified in Algorithm 2.

Figure 4.3 shows a comparison of our heuristic with the SJF scheduler. It is evident that a scheduler, which statically schedules all background jobs, always keeps the device busy. As a result, the user may experience delayed responses from the device. Our augmented scheduler conservatively uses the free computation cycles of the device such that foreground jobs have some buffer cycles left and therefore the user can use the device more comfortably. We can see from the figure that CPU utilization is not always 100% and has some room to cater to additional resources demanded by the foreground jobs and as a result the user is expected to experience a better response from the device. As expected, in the case when our scheduler is used, the background job takes longer time to complete.

4.5 Usage Pattern for Mobile Grid

Cloud computing involving mobile devices is an active area of research which deals with utilization of mobile devices in collaborative computing with cloud's back-end computing infrastructure. This paradigm of computing has two facets: in one, mobile devices are used as computing resources by the back-end infrastructure and, in the other, mobile devices use the back-end infrastructure to augment their computing capacity. Motivation for the first approach is the growing capacity and market for smart devices, which is stimulating the prospect of utilizing them as computing resources [20, 21]. Recent studies on the computing capacity of mobile devices claim that their computing power is comparable to that of desktops [2]. Frameworks like Hyrax [6], Serendipity [8] attempt to exploit the free computing capacity of mobile devices. The later one is a more traditional approach to mobile cloud computing (MCC) where devices utilize the computing power of back-end infrastructures by offloading some of its tasks to the back-end. This has led to several proposals of MCC for collaborative execution for executing compute-intensive work-flows [22–25]. The MCC paradigm has attracted considerable attention in both academia and the industry community in recent times. In this paper, we present a model and system for utilization of free computation cycles of mobile devices in MCC.

Several challenges remain to engage a mobile device as part of a computing infrastructure [26]. Some of these challenges are limited communication bandwidth, energy constraints, memory capacity, intermittent availability of resources like network or CPU, 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 smartphones among their senior employees [27]. 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, and privacy are mitigated. To encourage such an environment, the organization may as well provide incentives in suitable forms. In the company of the authors, reward 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 part of the paper, we adopt a simple localized mobile grid setting where the devices are accessible through a WiFi connection, and examine the problem of computation scheduling and workload management for improved timing/energy 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 need to be executed. The gateway device needs to decide on a schedule of computation and a selection mechanism so as to engage the mobile devices and utilize their donated computation cycles. The primary objective of driving this selection is to be able to finish execution of the application at the earliest possible time.

The gateway device is enabled with a task off-loader which is the controller of the task selection framework. When the off-loader wants to execute a task (in the form of a downloadable application), it invites bids from the owners of all devices connected to the off-loader. Additionally, the off-loader announces a deadline by which the computation has to finish. Associated with the task is a suitable reward to be earned by the winning bidder and also a penalty if the winner fails to deliver the task in time. Each owner, intending to participate in the bid, executes a pre-installed analysis agent on his device. The agent takes as input the advertised task and the deadline associated with the task, 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 off-loader 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 off-loader's challenge.

4.5.1 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, and 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 a 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.

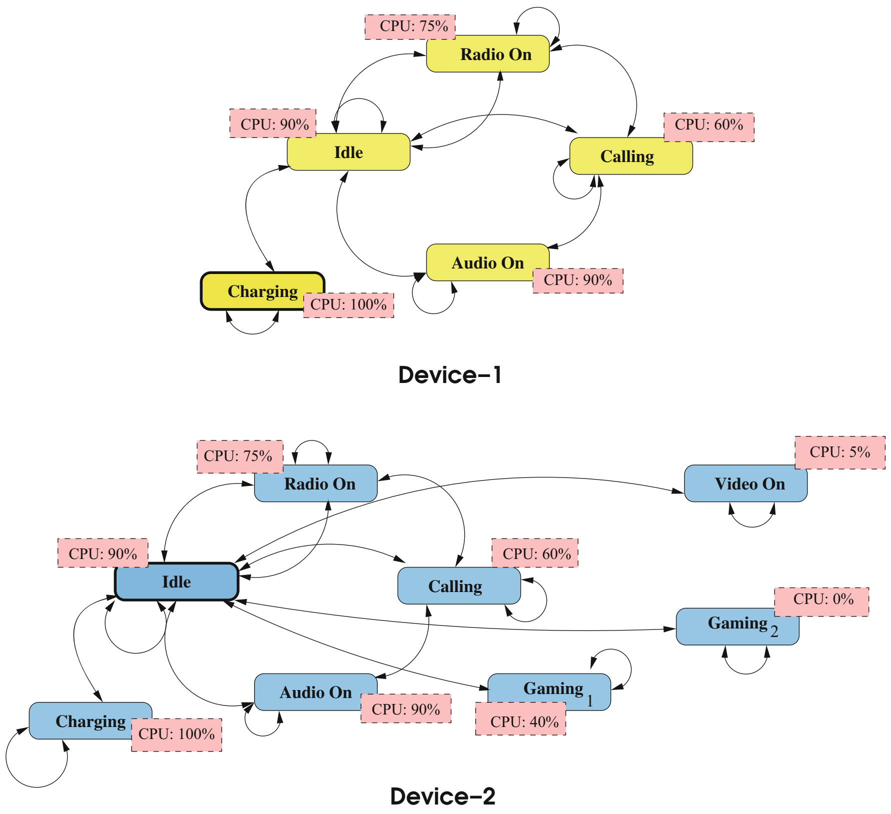


Fig. 4.4 Usage model with CPU availability

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 Fig. 4.4. 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 (Table 4.2).

Table 4.2 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	

Table 4.3 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	

4.5.1.1 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 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 *off-loader* awards the job to the one with earlier completion time, Device-1 is selected.

4.5.1.2 A More Realistic Scenario

In a more realistic setting, each device is expected to transition 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 4.3 shows the estimated completion times in this case and the *off-loader* 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 are 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 4.2. 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 [28] and a transition probability on each outgoing edge. AP implies the average time the device stays in the associated state. The revised model of the devices is depicted in Fig. 4.5. Now we apply the same

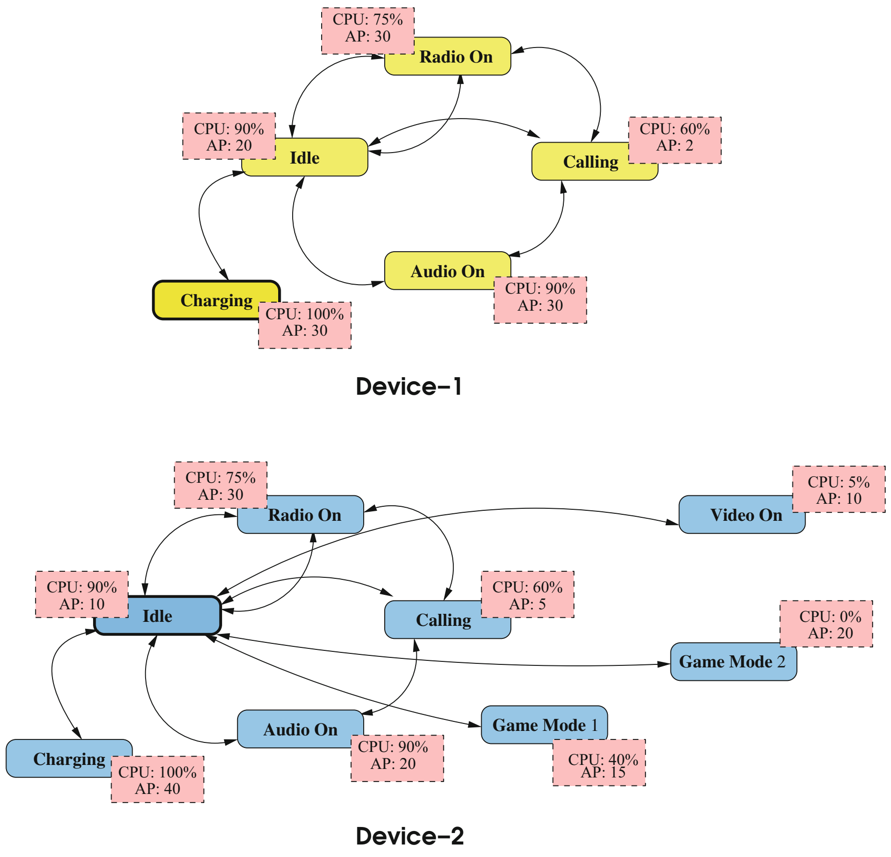


Fig. 4.5 Usage model with average permanence

Table 4.4 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 s	30 s	*idle*	90%	10 s	9 s
idle	90%	10.1 s	10 s	*charging*	100%	31 s	31 s
Completion time:		41.1 s		Completion time:		41 s	

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 4.4. The off-loader 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.

4.6 Bidding Methodology

The objective of the PFSM-AP described above is to characterize a device based on its free CPU cycles and the duration the device is likely to remain in a state, as depicted in the motivating examples in Sect. 4.5.1.

Given a mobile device D_i with a PFSM-AP model $\mathscr{U}_i =< \mathbf{S}_i, \mathscr{I}_i, \mathbf{T}_i, \lambda_i, \mathbf{H}_i, \mathbf{C}_i >$, a task J with its dataset and deadline Δ, the device needs to calculate its bid which can be presented to the off-loader 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:

- Examine all possible paths in the PFSM-AP graph from the state the device is in, at the time when bids are invited.
- Compute expected completion times on each of these paths considering that states in the path have different CPU availability.
- Exclude paths where the expected completion time is greater than the deadline.

- Exclude paths where the corresponding confidence value is less than some predetermined threshold.
- Determine a path which meets the deadline best and present the expected completion time on that path as the bid.

4.6.1 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

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

  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*:

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

Algorithm 3: Bid computation on a mobile device

```
   input : $J$ : The task to be executed along with dataset
   input : $\Delta$ : Deadline for the tasks
   input : $\mathbf{s}$ : Present state of the device $D_i$
   begin
1      Compute $w_i$ of $J$;
2      if $w_i > \Delta$ then No bid and return
3      $b_t \leftarrow \varnothing$ // best time
4      $\Pi_i \leftarrow$ paths $(\pi^i_j)$ on $\mathscr{U}_i$ satisfying C1 and C2;
5      for each path $p^i_j \in \Pi_i$ do
           if $\rho^i(p^i_j) < \Upsilon$ then continue $t \leftarrow \delta^i(p^i_j)$;
           if $(b_t > t)$ then $b_t \leftarrow t$
6      Bid with $b_t$;
```

where the term $\mathbf{C}(s^i_l) \times \mathbf{H}(s_l)$ denotes the quantum of computation done at the state s^i_l 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,

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

We modify the standard depth-first traversal [29] algorithm with constraints **C1** and **C2**, and also ignore self-loops involving a state with 0% CPU availability. These conditions bound the length of the paths (step 4 of Algorithm 3) to finite values since w_i is finite. Therefore, the algorithm terminates in 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 Sect. 4.5.1. Algorithm 3 uses a threshold Υ to filter out such paths.

4.7 Experiments with the Offloading System

To evaluate the proposed bidding-based offloading scheme, we built a small-scale cloud computing framework with an *off-loader* residing on a server and distributing jobs to a pool of mobile devices. We present some related results from the system. But, first we present our experiments with a simulation system.

We developed a simulation system to observe the behavior of our task offloading infrastructure. Experiments with our proposed job-offloading technique were carried out with the set of seven mobile devices described in Table 4.1. We simulated the *off-loader* system and also the task execution on the device VMs. Each VM simulates usage of the corresponding device by simulating the logged CPU loads for the last day in the trace file as discussed in Sect. 4.2.2. The simulated *off-loader* generated tasks of various kinds to be offloaded to these devices. When a task is awarded to a device, the winner device simulates the execution of the task while simulating the CPU load replayed from the trace file. For each task type, 100 similar tasks were generated and offloaded to devices based on bids. The number of tasks successfully completed on the devices (i.e., completed within the given deadline) is 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 *off-loader* generates jobs of various durations, assigns various deadlines to these jobs, invites bid, and submits the job to the winning device. Our job-offloading experiment was conducted for jobs whose execution time ranges from 4 to 29, and deadlines varying from $2\times$ to $10\times$ of the job execution time. 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 the given deadline. The result of the simulation experiment is shown in Fig. 4.6, where the horizontal axis represents variation in job execution duration and the vertical axis represents the offloading performance. 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 experiments that deadline is the most important factor for offloading tasks. When the deadline is very tight in comparison to the execution time of the task, task offloading is not beneficial. When the deadline is tight, if the device cannot operate with near 100% CPU availability all the time, the execution time is

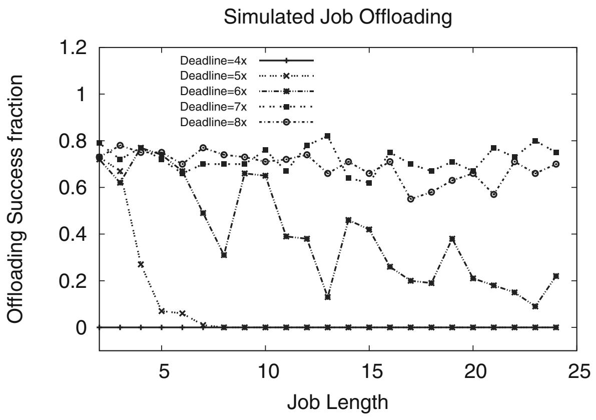


Fig. 4.6 Simulated system performance

more likely to overshoot the task deadline. When the deadline is very relaxed (e.g., approximately 6× that of job execution length) offloading technique works well and is beneficial. It is also evident from the result that very short tasks with very tight deadlines are not suitable to exploit this mobile computing system, rather a moderately high computation job with a relaxed deadline is more suitable from this framework. Another interesting observation is that for higher deadlines, offloading success does not improve at the same scale.

4.7.1 Working with Real Devices

In this phase, we evaluated the performance of our offloading system with an *off-loader* which has the seven devices, described earlier, for it to exploit. The details of the system are described below.

4.7.2 Architecture of Offloading System

The architecture of the system is presented in Fig. 4.7. The *off-loader* is composed of two principle components—(1) *Job Manager* which is responsible for managing incoming tasks. In our system setup, this module generates tasks to be scheduled in mobile devices. (2) *Bid Manager* is responsible to query available devices and then initiate bidding for each task, obtained from the job manager. The bid manager

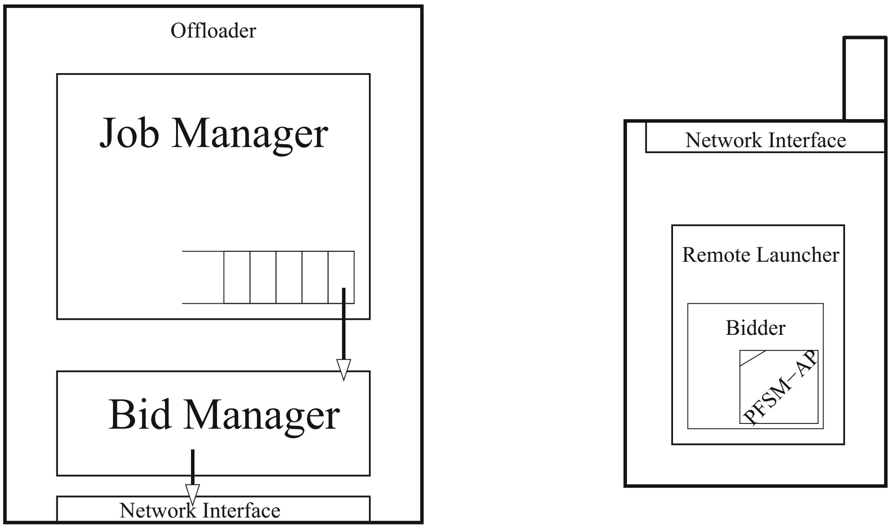


Fig. 4.7 Architecture of offloading system

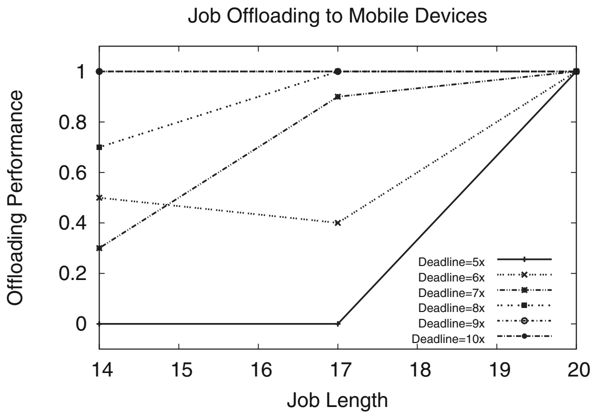


Fig. 4.8 Experimental system performance

creates a socket connected to a pre-defined IP of the server machine. Each of the mobile devices which volunteer to donate their CPU cycles need to install a lightweight android application, called *Remote Launcher*. The application connects and communicates with the *bid manager* of the *off-loader* through the pre-defined socket. The *bid manager* advertises a task on all the connections. The *remote launcher* computes its bid and sends back as a response to the *bid manager*. On award of a job, the *remote launcher* forks the job in local mobile devices, collects the result, and sends the data back to the *bid manager*. All the devices communicate using the WiFi network provided in our lab and the devices always remain within the communication range during the experiment.

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-value calculation. Figure 4.8 shows the job-offloading performance of the system. The result of this experiment shows that jobs with relaxed deadlines are good candidates for offloading.

4.8 Related Work

In recent times, there has been a significant volume of research on the theme of mobile cloud computing (MCC), which proposes the use of a collaborative computing infrastructure consisting of mobile devices and a back-end cloud computing system. MAUI [22] and CloneCloud [23] are the two notable systems which use a back-end computing infrastructure for collaborative job execution in mobile devices. Several other articles as well address the problem of work-flow partitioning in an MCC setting [30]. 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 work-flow with an objective of optimizing the cost (in terms of energy, time, and communication).

Authors in [31] address the problem of intermittent disconnection and analyze the same using a 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 [32]. 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 [33]. Systems like *Misco* [7] and *Hyrax* [6] extend Map-Reduce so that computation capabilities of mobile devices can be utilized. *Serendipity* is a task dissemination system over a mobile grid [8]. 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. Usage model-based task scheduling on mobile devices is attracting more attention in recent years. Authors in [3] present an analysis of the usage patterns of the communication module of a device. They define a state space composed of five states in which a device operates and present their analysis based on transitions among these states. Authors in [34] present a framework which analyzes pattern of usage of an application by its user and suggest a mechanism for optimizing launch time of the application. They claim that application launch latency is improved by this method which is reflected in improvement in user experience with the device.

4.9 Conclusion and Future Work

In this paper, we present a state transition-based system to model the usage pattern of a mobile device. The model captures the variation of free resources in the device based on its owner's usage pattern. We used this model to schedule background tasks in the device such that usage experience of the device does not degrade due to consumption of resources by the background tasks. We also used the same model in a different scenario of mobile cloud collaborative computing. We present a system, based on bidding, where mobile devices perform some tasks which a cloud computing infrastructure tends to offload to a set of participating mobile devices. Our usage pattern model, although being simple, can be effectively used in such diverse scenarios. Mechanisms for automatic learning of likelihood probabilities, using more advanced models for analysis, execution time estimation, designing more effective bidding, and 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

1. T. Danova, Gartner: mobile apps will have generated $77 billion in revenue by 2017 (2014), http://e.businessinsider.com/public/2373445
2. S. Sakr, Nvidia says Tegra-3 is a "PC-class CPU" (2011), http://engt.co/srvibU
3. J.-M. Kang, S.-s. Seo, J.-K. Hong, Usage pattern analysis of smartphones, in *2011 13th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (IEEE, Piscataway, 2011), pp. 1–8
4. J.-M. Kang, S.-s. Seo, J.W.-K. Hong, Personalized battery lifetime prediction for mobile devices based on usage patterns. J. Comput. Sci. Eng. (4), 338–345 (2011)
5. A. Shye, B. Scholbrock, G. Memik, P.A. Dinda, Characterizing and modeling user activity on smartphones: summary, in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1 (ACM, New York, 2010), pp. 375–376
6. E.E. Marinelli, Hyrax: cloud computing on mobile devices using MapReduce. Carnegie-Mellon Univ, School of Computer Science, Pittsburgh, PA, Tech. Rep. CMU-CS-09-164, Sept 2009
7. A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, V.H. Tuulos, Misco: a MapReduce framework for mobile systems, in *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments* (ACM, New York, 2010), p. 32
8. C. Shi, V. Lakafosis, M.H. Ammar, E.W. Zegura, 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, New York, 2012), pp. 145–154
9. A. Carroll, G. Heiser, An analysis of power consumption in a smartphone, in *USENIX Annual Technical Conference*, pp. 1–14 (2010)
10. L. Ardito, G. Procaccianti, M. Torchiano, G. Migliore, Profiling power consumption on mobile devices, in *ENERGY 2013, The 3rd International Conference on Smart Grids, Green Communications and IT Energy-Aware Technologies*, pp. 101–106 (2013)
11. A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, V.N. Padmanabhan, Bartendr: a practical approach to energy-aware cellular data scheduling, in *Proceedings of the 16th Annual International Conference on Mobile Computing & Networking* (ACM, New York, 2010), pp. 85–96
12. A. Chakraborty, V. Navda, V.N. Padmanabhan, R. Ramjee, Coordinating cellular background transfers using loadsense, in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking* (ACM, New York, 2013), pp. 63–74
13. P.K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V.N. Padmanabhan, G. Varghese, Radiojockey: mining program execution to optimize cellular radio usage, in *Proceedings of the 18th Annual International Conference on Mobile Computing & Networking* (ACM, New York, 2012), pp. 101–112
14. A. Shye, B. Scholbrock, G. Memik, Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures, in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (ACM, New York, 2009), pp. 168–178
15. V. Tiwari, S. Malik, A. Wolfe, M.T.-C. Lee, Instruction level power analysis and optimization of software, in *Technologies for Wireless Computing* (Springer, New York, 1996), pp. 139–154
16. S. Hao, D. Li, W.G. Halfond, R. Govindan, Estimating mobile application energy consumption using program analysis, in *2013 35th International Conference Software Engineering (ICSE)* (IEEE, Piscataway, 2013), pp. 92–101
17. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra et al., The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. (TECS) **7**(3), 36 (2008)
18. M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling. Math. Oper. Res. **1**(2), 117–129 (1976)

19. D.G. Feitelson, Job scheduling in multiprogrammed parallel systems: extended version, IBM research RPT. RC **19790**, 87657 (1979)
20. F. Bonomi, R. Milito, J. Zhu, S. Addepalli, 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, New York, 2012), pp. 13–16
21. A. Mukherjee, H.S. Paul, S. Dey, A. Banerjee, Angels for distributed analytics in IOT, in *2014 IEEE World Forum on Internet of Things (WF-IoT)* (IEEE, Piscataway, 2014), pp. 565–570
22. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (ACM, New York, 2010), pp. 49–62
23. B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: elastic execution between mobile device and cloud, in *Proceedings of the Sixth Conference on Computer Systems* (ACM, New York, 2011), pp. 301–314
24. S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in *INFOCOM, 2012 Proceedings IEEE* (IEEE, Piscataway, 2012), pp. 945–953
25. M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z.M. Mao, X. Chen, Comet: code offload by migrating execution transparently, in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, OSDI*, vol. 12, pp. 93–106 (2012)
26. T. Phan, L. Huang, C. Dulan, Challenge: integrating mobile wireless devices into the computational grid, in *MobiCom '02: Proceedings of the 8th Annual International Conference on Mobile Computing and Networking, MOBICOM-2002*, pp. 271–278 (2002)
27. A. Agarwal, Enterprise smartphone usage trends (2011), http://bit.ly/loIqE1
28. X. Li, A. Gray, D. Jiang, X. Mao, Sufficient and necessary conditions of stochastic permanence and extinction for stochastic logistic populations under regime switching. J. Math. Anal. Appl. **376**(1), 11–28 (2011)
29. R. Tarjan, Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
30. M.R. Rahimi, N. Venkatasubramanian, A.V. Vasilakos, 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, 2013), pp. 75–82
31. S.-M. Park, Y.-B. Ko, J.-H. Kim, Disconnected operation service in mobile grid computing, in *Service-Oriented Computing-ICSOC 2003* (Springer, New York, 2003), pp. 499–513
32. E. Jung, F. Maker, T.L. Cheung, X. Liu, V. Akella, Markov decision process (MDP) framework for software power optimization using call profiles on mobile phones. Design Autom. Embed. Syst. **14**(2), 131–159 (2010)
33. H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches. Wirel. Commun. Mob. Comput. (2011)
34. W. Song, Y. Kim, H. Kim, J. Lim, J. Kim, Personalized optimization for android smartphones. ACM Trans. Embed. Comput. Syst. (TECS) **13**(2s), 60 (2014)