$MobiDiC$: Exploiting the Untapped Potential of Mobile Distributed Computing via Approximation

Parul Pandey and Dario Pompili

Department of Electrical and Computer Engineering Rutgers University–New Brunswick, NJ, USA

E-mails: {parul_pandey, pompili}@cac.rutgers.edu

***Abstract*—Mobile computing is one of the largest untapped reservoirs in today's pervasive computing world as it has the potential to enable a variety of in-situ, real-time applications. Yet, this computing paradigm suffers when the available resources—such as device battery, CPU cycles, memory, I/O data rate—are limited. In this paper, the new paradigm of *approximate computing* is proposed to harness such potential and to enable real-time computation-intensive mobile applications in resource-limited and uncertain environments. A reduction in time and energy consumed by an application is obtained via approximate computing by decreasing the amount of computation needed by different tasks in an application; such improvement, however, comes with the potential loss in accuracy. Hence, a Mobile Distributed Computing framework, $MobiDiC$, is introduced to determine *offline* the 'approximable' tasks in an application and a light-weight algorithm is devised to select the approximate version of the tasks in an application during run-time. The effectiveness of the proposed approach is validated through extensive simulation and testbed experiments by comparing approximate versus exact-computation performance.**

***Index Terms*—Mobile device clouds; Approximate computing; Mobile perception application; Workflows.**

I. Introduction

Vision: Technology has the power to adapt to the limitations of human perceptions. With high-speed and time-lapse photography, we can appreciate and understand processes not visible to human eye (as either happening too fast or too slowly); with the creation of overlays from multiple, spatially separated data sources on Google Earth we can visualize information not naturally visible to human senses; with deep-learning techniques we can achieve leaps of improvement in mature domains such as speech recognition [1]. All these technologies, which help us understand phenomena unimaginable otherwise, have *computation as their core infrastructure*. We envision mobile computing to become pervasive and bring all these technologies anywhere and everywhere!

Motivation: The state of the art in mobile computing falls short in achieving this vision on hand-held devices. This computing paradigm, in fact, suffers when the available resources—such as device battery, CPU cycles, memory, I/O data rate—are limited. Considering the slow performance improvement in mobile-device architecture and battery, it is unlikely that the fundamental problems limiting a faster trend will be solved in the near future. In spite of these limitations, many computation-intensive applications from a variety of domains such as computer vision (e.g., object recognition, panorama stitching), machine learning (e.g., natural language translators, speech recognizers), and artificial intelligence (e.g., gaming applications, online learning) are expected to work seamlessly on smart hand-held devices and give results in *real time*. Work on mobile cloud computing [2], [3] has been done whereby the application execution is moved from the resource-constrained mobile devices to powerful and centralized remote computing platforms such as the Cloud. However, good connectivity from the device to a WiFi network may not always be possible. Although 3G has a near-ubiquitous coverage, recent studies have shown that round-trip times are often long and that communication links are bandwidth limited; the former have been shown to be consistently on the order of hundreds of milliseconds and in some cases even reaching seconds [4]. This is unacceptable in real-time/interactive applications, which require low response times.

Our approach: We present $MobiDiC$, an "energy-" and "accuracy-aware" framework that exploits the new paradigm of *approximate computing* to enable near real-time mobile applications in resource-constrained environments. Approximate computing reduces the amount of computation that an application is expected to perform, as a result of which the execution time, i.e., the *makespan*, as well as the energy consumption reduce. The gain achieved via reduction in makespan and energy expenditure, however, comes with a potential loss in the accuracy of the results (within acceptable limits). We introduce reduction in computational cost via two transformations—namely, *substitution* and *discarding*—both of which can be applied to the *tasks* in an application, where each task is constituted by a subroutine/function along with a set of input parameters. These transformations enable the paradigm of approximate computing via the *joint* optimization of function and parameter space of an application.

Our approximate-computing framework consists of an *offline* and *online* phase (as shown in Fig. 1). In the offline phase, we introduce a powerful workflow representation scheme to determine which tasks in the application can be approximated; we also provide statistical guarantees on the reduction in makespan achieved by varying the application acceptable accuracy loss bound. The online phase is executed at run-time and leverages the information obtained from the offline phase to determine the accuracy loss to be incurred in order to meet the application deadline given the computational capabilities of the device. In this paper, we propose a light-weight probabilistic algorithm to select approximated tasks that are most likely to meet the application deadline within the estimated accuracy

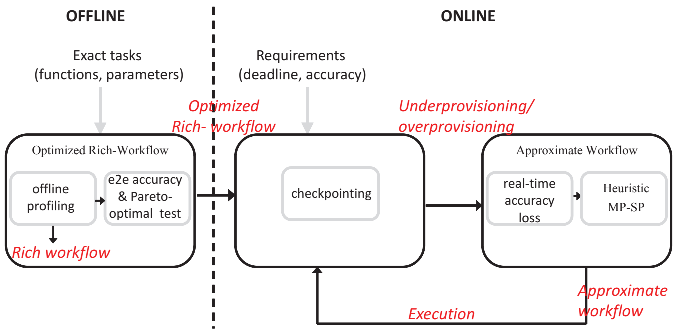


Fig. 1: Block diagram to represent $MobiDiC$ approximate computing framework. The offline phase determines the task(s) in an application which can be approximated. This information is leveraged at run-time of the application along with application deadline and acceptable accuracy loss bound.

loss bound and under run-time uncertainties.

We motivate and study the performance of approximate computing via two well-known and broadly-applied recognition algorithms, namely, Canny edge detection [5] and the Scale Invariant Feature Transform (SIFT) [6]. Our results show that an approximate implementation may perform significantly better than the exact implementation of suboptimal algorithms. We observed that when approximate computing is applied the execution time decreases up to 40% at the price of only 5% in loss of accuracy. We also present results showing the performance of approximate computing in an uncertain distributed mobile environment via our experimental testbed.

Contribution: The following are our main contributions.

- $MobiDiC$, an energy- and accuracy-aware approximate-computing framework to support real-time mobile applications in limited resource environments.
- An online algorithm that selects the approximated tasks that should be executed to meet the application deadline under uncertainties encountered at run-time.
- Validation of our approach through simulation and testbed experiments comparing the performance of approximate versus exact computing.

Paper outline: The remainder of this paper is organized as follows. In Sect. II, we review the state of the art in traditional mobile computing and approximate computing. In Sect. III, we introduce the entities of our approximate-computing framework. In Sect. IV, we discuss how approximate computing can be applied to time-critical applications during run-time. In Sect. V, we provide details of our experimental setup and study the performance of approximate versus exact computing. Finally, in Sect. VI, we conclude the paper.

II. Related Work

We briefly review the state of the art in the area of mobile computing and approximate computing. We explain the limitations of these approaches and how our work differs from them. Ours is the first work where the paradigm of approximate computing is exploited to enable real-time applications in mobile computing space.

Traditional mobile computing: Much work has been done in the area of mobile computing with a focus on enabling mobile applications in resource-limited environments. In mobile cloud computing, researchers have focused on augmenting the capabilities of mobile devices in the field by offloading costly (compute and energy-intensive) tasks to dedicated wired-grid [7] or cloud resources in a transparent manner. However, these approaches are not suitable for enabling data-intensive applications in real time due to prohibitive communication cost and response time, significant energy footprint, and the curse of extreme centralization. To circumvent these challenges, mobile device cloud has been introduced [8]. This paradigm was based on splitting the tasks in an application and executing them in parallel on nearby mobile devices. However, the local resource pool may suffer from scarcity of devices which computation can be outsourced to, or from uncertain network connectivity and device availability.

Approximate computing: Researchers have developed energy-aware programming languages by introducing approximation at different levels such as mathematical operations and storage of data structures (in the form of unreliable register, data cache, and main memory). One such language is EnerJ [9], which allows the programmer to annotate data as ‘approximate’ or ‘precise’. The system then automatically maps approximate variables to low-power storage, uses low-power operations, and applies more energy-efficient algorithms provided by the programmer. In [10], [11], [12], the authors employ various approximation techniques such as loop perforation and multiple implementations of tasks. Our work, on the other hand, jointly applies different approximation techniques to both tasks and input parameters of the application. Our novel solution handles the uncertainties arising at run-time. It also estimates the accuracy loss that should be incurred—based on the resource availability and application deadline—and approximates the tasks in such a way as to meet the accuracy loss bound.

III. $MobiDiC$—Approximate Computing Framework

Our goal is to achieve dynamically a *tradeoff* between *accuracy* (or optimality of the results produced by an application) and *utilization* of the available resources (such as battery, CPU cycles, memory, and I/O data rate). We first discuss a structural approach to approximation in mobile computing. Then, we present the approximation techniques that can be applied to different tasks in an application. We now define an offline phase that helps us identify promising applications whose tasks can be approximated so to gain significant benefits in energy at the cost of marginal loss in accuracy.

A. Ontology of Approximation

Types of tasks: An application consists of the execution of a set of tasks to obtain the required result. We consider a task in an application to be “elementary" if it cannot be split further into sub-tasks. Each task is represented by an executable code/function (to represent a functionality that cannot be split further) and a set of input parameters. We divide tasks into two different categories, namely, *approximable* and *non-approximable*. We assume that the information about the type of task is provided by the application developer or via offline profiling (discussed later).

Approximable: Tasks that can be approximated to achieve significant savings in energy and/or execution time, with however a potential loss of accuracy in the result.

Non-approximable: Tasks whose execution without any approximation is necessary for the success of the application, i.e., if any approximation technique were applied on these tasks, the application would not generate meaningful results.

Types of approximations: We introduce approximation through two transformations, namely *substitution* and *discarding*, which are applied to different tasks (both at function and input parameter) of the application. Specifically, the former transforms the task(s) in exact computation with those with lower degree of complexity; whereas the latter involves removing certain task(s) of an application used for exact computation. We now briefly explain these transformations.

Substitution: This transformation requires substitution of a computation task (its execution code or input parameter) by a simpler task. At the *function level*, this operation refers to the substitution of a task in exact computation by a computationally less-demanding task with potential loss in accuracy. This requires the availability of multiple implementations of a task, each with a different degree of complexity (e.g., 2D Gaussian function serves as a filtering kernel in image processing; however, it can be replaced with recursive Gaussian or box filters, which are both computationally much less demanding albeit they provide lower accuracy [13]). This transformation requires domain knowledge.

At the *parameter level*, it refers to the scaling up or down of the exact implementation value of a task parameter. A *substitution factor* f determines the factor by which the value of the approximate parameter varies with respect to (w.r.t.) the exact parameter; for example, if the value of the parameter in the case of exact computation is p, the new value via substitution will be $p * f$. For example, in Content Based Image Retrieval (CBIR) applications, whose aim is to retrieve image features via histogram analysis, the number of bins can be decreased (here, $f < 1$) in such a way as to reduce the computational cost at the cost of a decreased output accuracy.

Discarding: Applications consist of tasks that successively improve upon the results obtained from previously executed tasks. Discarding transformation involves not executing these tasks so to reduce energy consumption at the cost, however, of reduced accuracy. At the *function level*, if the user-specified accuracy is achieved by a subset of the tasks, the application can choose to skip the remaining task and terminate early; hence, discarding certain redundant tasks can lead to significant benefits in terms of energy and/or execution time.

At the *parameter level*, it refers to early termination or skipping of number of iterations in a task. Skipping parameter space was introduced in [10], where one of every n scheduled iterations was executed, as a result of which the systems performs fewer computations than its exact-implementation counterpart. Discarding transformation can be applied to traditional Fast Fourier Transform (FFT)-based algorithms to get suboptimal results with reduced computation cost [14].

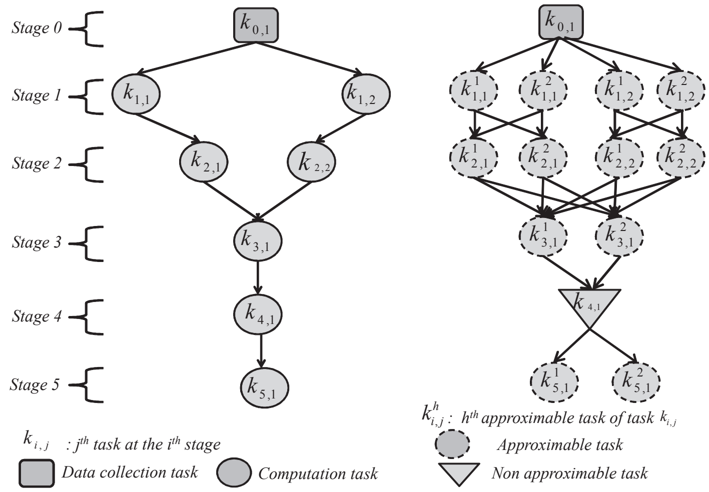


Fig. 2: (Left) Exact workflow representation; (Right) Rich workflow constructed by extending exact workflow to represent approximation transformations. *Substitution* transformation is represented by multiple (alternate) tasks in a stage (e.g., tasks $k^1_{1,1}$, $k^2_{1,1}$, are approximable tasks for task $k_{1,1}$ in Stage 1); *Discarding* transformation is shown by skipping a task in a certain stage (e.g., Task $k_{2,2}$ in the exact workflow is skipped in Stage 2 and $k^2_{3,1}$ is executed immediately after $k^2_{1,2}$).

Accuracy metric: In our framework we compare the accuracy or quality of output of an application by executing the application via exact computation and by applying the aforementioned approximate techniques. Different metrics such as F_1 *Score* (i.e., $2\frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$), *peak-signal-to-noise ratio* or any other application-domain metrics can be used to measure the output accuracy. An exact-computation implementation gives the highest accuracy achievable for that application. The percentage loss in accuracy of the output when applying approximation w.r.t. exact computation is calculated as $T = \frac{Q-\hat{Q}}{Q} \cdot 100$, where Q is the accuracy of the output obtained by exact implementation of the application and $\hat{Q}$ is the accuracy of the output obtained by the approximate implementation of the application.

B. Transformation of Workflows

The order of execution of multiple tasks in an application can be specified by a *workflow*. Here, we first explain our workflow representation for an exact computation implementation, and then show how such workflow is transformed for an approximate-computation implementation. Transformation of workflows is accomplished offline and is leveraged at runtime to make decisions when the application is executed.

Exact-workflow representation: Let the exact workflow $G(V, E)$ be presented by a Directed Acyclic Graph (DAG), as shown in Fig. 2(Left). The workflow is composed of multiple stages with a set of tasks to be performed at each stage. It is a graphical representation of the set of tasks, $V = \{k_{i,j}\}$, where $k_{i,j}$ is the j^{th} task in the i^{th} stage. The edges in the workflow indicate the dependencies between tasks. Tasks at a stage cannot be executed unless all the tasks in the previous stage have been completed as tasks at a stage accept data from the previous stages. In the workflow representation, square

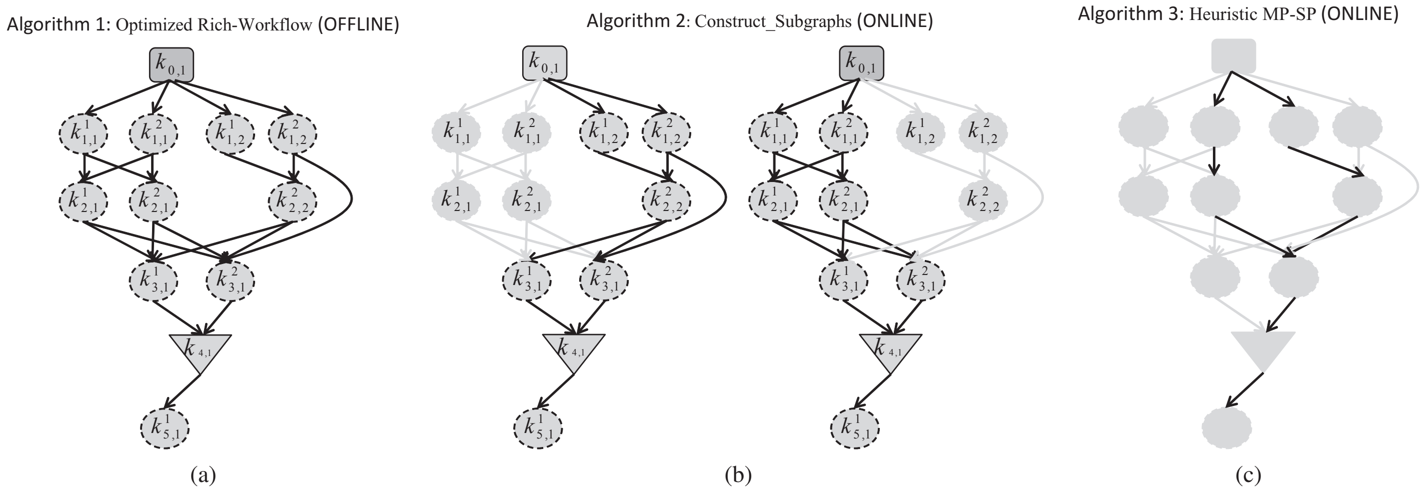


Fig. 3: Illustration of (a) *Optimized Rich-Workflow* constructed from Rich Workflow (Fig. 2 (Right)) by reducing the task space; (b) *Subgraphs* formed for multiple independent tasks of the optimized workflow. The subgraphs are created by Algorithm 2: $\mathrm{Construct\_Subgraphs}$ and are executed only when the exact workflow is task-parallel with multiple independent tasks at different stages of the workflow; (c) *Approximate workflow* extracted from Optimized Rich-workflow at run-time via Algorithm 3: $\mathrm{Heuristic\ MP-SP}$.

nodes (□) represent the input data whereas circular nodes (◯) represent the computation tasks.

Determining approximable tasks: We explain now how to identify approximable tasks in an application. For example, to determine if Task $k_{1,1}$ is approximable, we first apply discarding transformation separately to each of its input parameter and alternate functions available. We repeat the same by using substitution transformation. Such procedure results in multiple approximate versions of the task. Then, we replace Task $k_{1,1}$ with one of its approximate versions while all other tasks in the exact workflow are left unchanged. After such replacement, we calculate the resulting *makespan* and accuracy (Q) of the workflow. The exact-computation implementation gives the highest accuracy results for the application. The speed-up (*sp*) obtained from one of the approximate versions is calculated by dividing the makespan of the approximate version by the makespan associated with its exact implementation. This is done for a large number of input data so to get the average speed up ($\overline{sp}$) and average accuracy ($\overline{Q}$).

If any approximate version of Task $k_{1,1}$ provides $\overline{sp} > 1$ along with accuracy loss less than the acceptable loss T_A, then $k_{1,1}$ is considered an approximable task. The approximate versions that do not satisfy these constraints are discarded. If none of the approximate versions of a task satisfies these constraints, then that task is deemed non-approximable. If multiple implementations of a task are available, then substitution transformation can be applied; otherwise, only discarding transformation is performed.

Rich-workflow representation: An *approximate instance* of an exact workflow is the one whose tasks satisfy the constraints mentioned earlier. Collection of all the approximate instances of an application forms a rich-workflow, $G^R(V^R, E^R)$. In Fig. 2(Right), we can see that each approximable task ($k_{i,j}$) in the exact workflow has a corresponding approximate version ($k^l_{i,j}$). Each approximate version ($k^l_{i,j}$) in the exact workflow is selected via Algorithm 1 when $\overline{sp} > 1$ and $\frac{|\overline{Q}-\overline{\hat{Q}}|}{\overline{Q}} \cdot 100 < T_A$.

Algorithm 1: $\mathrm{Optimized\ Rich-Workflow}$ (Offline)

```
Input: $A$-Application, $\mathcal{B}$-set of approximate versions of all tasks in $A$,
       $\mathcal{I}$-Test data set, $T_A$-acceptable accuracy loss of $A$
Output: $G^O(V^O, E^O)$-Optimized rich-workflow
$\hat{\mathcal{B}} = \emptyset$;
for $b \in |\mathcal{B}|$ do
    for $i \in \mathcal{I}$ do
        Replace an exact task in $A$ with $b$;
        Execute $A$ with input $i$ to get $\hat{Q}_i$ and $sp_i$ ;
    end
    $\overline{\hat{Q}} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \hat{Q}_i$ , $\overline{sp} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} sp_i$ ;
    if $\frac{|\overline{Q}-\overline{\hat{Q}}|}{\overline{Q}} \cdot 100 < T_A \wedge \overline{sp} > 1$ then
        $\hat{\mathcal{B}} = \hat{\mathcal{B}} \cup b$;
    end
end
Construct Rich-workflow using tasks in $\hat{\mathcal{B}}$ ;
Select Pareto-optimal approximate instances to form the Optimized
Rich-workflow;
```

The edge of the rich-workflow is represented as $e^{m,n,l}_{i,j,h} = \{< k^h_{i,j}, k^l_{m,n} > \in E^R\}$. Note that, in Fig. 2(Right), non-approximable tasks are represented by triangular nodes (∇).

Reducing approximation space: We discard the approximate instances in the rich workflow that give accuracy loss less than T_A. To further reduce the approximation space in the rich workflow, we select only those approximate instances of the application that are *Pareto Optimal*. An approximate instance is Pareto-optimal if there is no other approximate version of that task that provides *both* better speed up *and* accuracy, i.e., t_1 is a Pareto-optimal approximate instance iff there is not any other approximate instance t_2 s.t. $\overline{\hat{Q}}(t_1) \leq \overline{\hat{Q}}(t_2) \wedge \overline{sp}(t_1) \leq \overline{sp}(t_2)$, where *at least one* of the inequality is *strict*. The collection of these approximate instances, which consist of Pareto-optimal approximate instances that give percentage accuracy loss w.r.t. exact computation less than T_A, form an *optimized rich-workflow*, i.e., $G^O(V^O, E^O)$. Figure 3(a) is an example of optimized workflow formed by applying Pareto-optimal test on Fig. 2(Right).

Need for an offline phase: The offline tools mentioned above help the programmer identify the functions and input parameters of the application that can benefit from various approximation techniques. However, these tools are too heavy to be used during run-time, as the cost of executing these tools at run-time may be greater than the savings in time and energy obtained from approximation of the application. As a result, these tools are implemented only offline. Selection of Pareto-optimal tasks reduces the complexity of online mechanisms as it reduces the approximation space and helps the application select optimal approximated tasks from a much smaller space as well as meet the deadline constraints.

IV. Real-time approximate computing

Uncertainty at run-time arises when the execution time of the application during run-time does not mirror the behavior observed during the offline profiling. Execution time of tasks depends on its implementation along with input parameters, size of input data, input value, and architecture of the execution location. For a given implementation of a task and input parameter value, the task execution time can vary significantly with input data; in certain situations, it can lead to missing the application deadline. In order to enable approximate computation at run-time and get results in near real time, we should be able to answer the following questions:

- Given the resources available, how much accuracy loss should be incurred to provide meaningful results within the application deadline?
- Which tasks should be executed to deliver results within the acceptable accuracy loss while simultaneously meeting such deadline?
- How does the uncertainty in the mobile distributed environment impact the performance gain of approximate computing?

Determination of accuracy loss: Let sp be the amount of speed up required to complete the execution of the application within its specified deadline. Our goal is to specify to the user at run-time how much accuracy loss needs to be incurred in order to achieve this speed up, given the available computational resources. For this we fit the offline profiling data of the Canny edge-detection application (black circles) with a non-linear model, $\mathrm{A} \cdot \exp(\frac{\mathrm{sp}}{\mathrm{B}})$, which is shown by red-dotted line in Fig 4. Goodness of fit statistics such as root mean square error are used to estimate the coefficients A and B.

Construction of approximate workflow: Our next goal is to determine the approximate instance of the optimized workflow that meets both the makespan and the estimated accuracy loss bound. Such approximate instance is called an *approximate workflow*. We now present a light-weight solution to determine such approximate workflow at run-time by leveraging the results from offline profiling.

Each edge, $e^{m,n,l}_{i,j,h}$, of the optimized rich-workflow gives the value of execution time of task $k^{l}_{m,n}$, denoted as $d(e^{m,n,l}_{i,j,h})$, after task $k^{h}_{i,j}$ has been executed. For a particular device, the offline profiling provides us with the execution time for running a task of an application with different input data, resulting in varying execution times for the task. Hence, the execution time of an edge can be defined as a real-valued random variable in (0, $+\infty$) varying with the input data set. Theoretically, the distribution of $d(e)$ for any edge e can be captured by a Probability Density Function (PDF); however, in reality, the PDF, $f_d(e)$, of $d(e)$ is often unknown. Instead, a set of samples $\hat{d}(e) = [d_1, d_2, \ldots d_W]$, which are obtained from offline profiling, are used to approximate the distribution of $d(e)$, where W is the number of trials in the offline profiling. Each sample of $\hat{d}(e)$ has a $\Pr\{\hat{d}(e) = d_w\} \in (0, 1]$, where $\sum\limits_{w=1}^{W} \Pr\{\hat{d}(e) = d_w\} = 1$. For sake of compactness, we simply denote $\hat{d}(e)$ as $d(e)$.

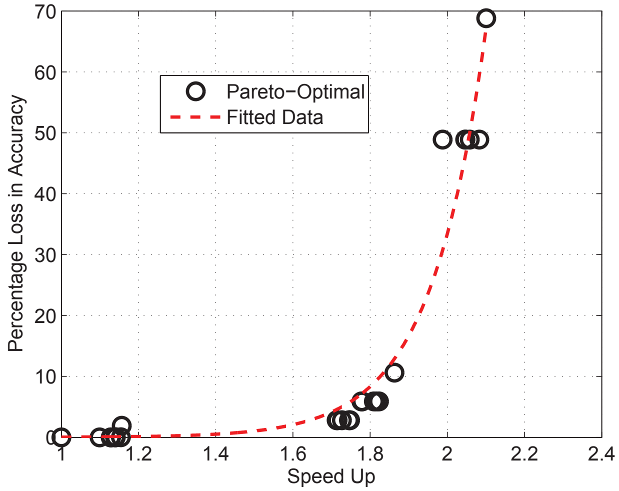


Fig. 4: Fitting offline profiling data with a non-linear model, $\mathrm{A} \cdot \exp(\frac{\mathrm{sp}}{\mathrm{B}})$, to estimate at run-time the loss in accuracy that should be incurred to achieve a certain speed up.

A path is a set of consecutive edges that connect the source (first task in the workflow) to the destination node (terminal task in the workflow). The execution time (or makespan) of an application is the sum of execution times of all the edges in a path p and is given by $D(p) = \sum\limits_{e^{m,n,l}_{i,j,h} \in p} d(e^{m,n,l}_{i,j,h})$. As $d(e^{m,n,l}_{i,j,h})$ is a random variable, $D(p)$ is also a random variable. The delay of a sample path, w, of $D(p)$, associated with a single trial (i.e., an input data) is given by $\sum\limits_{\substack{e^{m,n,l}_{i,j,h} \in p \\ w \in \mathrm{W}}} d_w(e^{m,n,l}_{i,j,h})$. Each edge is associated with W instances and there are multiple paths in an application. Our goal is to create a light-weight run-time algorithm; hence, we reduce the complexity of the problem by transforming each edge $d(e)$. We find the edge sample w that has the highest probability, i.e., $w := \max p_w(e), \forall e$, and substitute $d(e)$ with $d_w(e)$. Given an application deadline M, our goal is to find a path $p^* = \arg\max_p \Pr\{D(p) \leq M\}$, such that, for every other path p, the following holds,

$$\Pr\{D(p^*) \leq M\} \geq \Pr\{D(p) \leq M\}, \forall p. \qquad (1)$$

The probability of a path is given as the product of the probability of edges on that path. To solve this problem, we transform each probability function as a cost function

TABLE I: Characteristics of the computing devices in our testbed.

Devices	Samsung Galaxy Tab	ZTE Avid N9120	Huawei M931	Toshiba Satellite	Dell Inspiron	Acer Asprire
Type of devices	Tablet	Smartphone	Smartphone	Laptop	Netbook	Netbook
No. of devices	2	3	1	1	1	1
CPU	1GHz Dual-core ARM	1.2GHz Dual-core	1.5GHz Dual-core	2.13 GHz i3 Intel	1.66 GHz N450 Intel	1.60 GHz N270 Intel
OS	Android v4.0	Android v4.0	Android v4.0	Windows 7	Windows 7	Windows XP
RAM [GB]	1	0.512	1	4	1	2
Battery [mAh]/[V]	7,000/4	1,730/5	1,650/10.8	4,200/10.8	5,200/11.1	4,840/11.1

Algorithm 2: Construct_Subgraphs (Online)

```
Input: $G^O(V^O, E^O)$
Output: $G_{sub}$- Subgraphs
Child set: $Child \leftarrow \emptyset$ ;
$\mathcal{I}$ contains all $i^{th}$ stages, where, $j > 1$ ;
for $i' \in \mathcal{I}$ do
    $J = \max j$ for $i'^{th}$ stage;
    while $j' > J$ do
        $i\_temp = i'$ ;
        $G_{sub}(i', j') = G_{sub}(i', j') \cap k^h_{i\_temp, j'} \forall h$ ;
        $i\_temp = i\_temp + 1$ ;
        if $i\_temp \notin \mathcal{I}$ then
            $G_{sub}(i', j') = G_{sub}(i', j') \cap Child(k^h_{i\_temp,1})$;
        end
        else
            break ;
        end
    end
end
```

Algorithm 3: Heuristic MP − SP (Online)

```
Input: $G^O(V^O, E^O)$, $K$, $M$, $child\_val(v)$-number of children of
       node $v$, $d_h(e)$, $c_h(e) \forall \{e, h\}$
Output: $G^A(V^A, E^A)$- Approximate workflow
$count \leftarrow 1$;
$d(e) \leftarrow d_w(e)$ & $c(e) \leftarrow c_w(e)$, where $w := \min c_w(e) \; \forall \; e$;
while $G^O(V^O, E^O) \neq \emptyset \cup count < K$ do
    $[P, D] = Dijkstra(G^O(V^O, E^O))$ ;
    for $v \in P$ do
        $child\_val(v) = child\_val(v) - 1$ ;
    end
    if $D > M$ then
        break ;
    end
    else
        $G^A(V^A, E^A) \leftarrow G^A(V^A, E^A) \cup P$ ;
        Remove tasks from $G^O(V^O, E^O)$ with $child\_val = 0$;
        $count = count + 1$;
    end
end
```

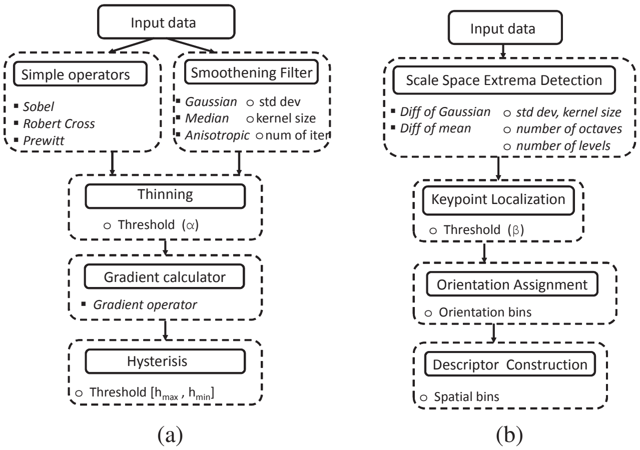


Fig. 5: Block diagram showing different tasks and parameters for object recognition using (a) *Canny edge detection* and (b) *Scale Invariant Feature Transform (SIFT)*. Each dashed block contains multiple implementations of approximable task types (with varying degree of complexities) and parameters.

$c() = -\log(f(d(e)))$, which makes the components additive. Now, as the probability associated with an edge increases, the cost decreases; hence, *our goal is to find the path, with the lowest cost function, that simultaneously meets the makespan constraints*. We formulate this problem as a *Restricted Shortest Path (RSP) Problem*. Given a network $G^O(V^O, E^O)$, execution time and cost associated with each edge in E, and application deadline M, our goal is to find a path (p^*) that solves the following problem,

$$\min \sum_{e^{m,n,l}_{i,j,h} \in p^*} c(e^{m,n,l}_{i,j,h}), \quad s.t. \sum_{e^{m,n,l}_{i,j,h} \in p^*} d(e^{m,n,l}_{i,j,h}) \leq M. \quad (2)$$

If our application is task parallel with independent parallel tasks at certain stages, then we first construct subgraphs within the optimized workflow such that, in each subgraphs, there is only one task per stage. Algorithm 2, illustrated in Fig. 3(b), shows our proposed algorithm to construct subgraphs with one independent task per stage for task-parallel workflows. Algorithm 3, illustrated in Fig. 3(c), shows our proposed heuristic to solve the restricted shortest path problem presented above and extracts the approximate workflow.

V. Performance Evaluation

This section is geared towards quantifying the gain of approximate over exact computing to support various computer-vision algorithms. We present the results from offline profiling by giving statistical bounds on the speed up achieved via approximate computing along with the accuracy loss incurred.

Experimental testbed: We present the various elements of our experimental testbed, which is shown in Table I. Our testbed comprises of state-of-the-art, heterogeneous computing

TABLE II: Number of approximate instances selected in various stages of the offline profiling for Canny edge-detection algorithm.

Accuracy Loss [%]	All Workflows	Rich Workflows	Optimized Workflows	Discarded
20	150	55	18	130
40	150	59	18	126
65	150	82	23	98
80	150	97	24	82
100	150	97	24	82

TABLE III: Gain achieved by approximating tasks of Canny edge detection and SIFT.

	Function/Parameter	Range	Accuracy Loss [%]	Speed Up
Canny	Threshold	[0,1)	2.76	1.75 ± 0.01
	Sigma	[0,1]	0.01	1.14 ± 0.01
	Kernel Size	[3:11]	7.04	1.13 ± 0.012
SIFT	No. of Octaves	[1,10]	0.7 ± 0.05	1.5 ± 0.022
	No. of Spatial Bins	[1,10]	0.8 ± 0.05	2.0 ± 0.025
	No. of Orientation	[1, 2^3]	0.6 ± 0.05	1.5 ± 0.034
	No. of Level	[1,10]	0.85 ± 0.06	2.0 ± 0.025

devices (tablets, smartphones, laptops, and notebooks) that vary by type of device, platform, RAM, and processing power.

Applications implemented: We motivate and study the performance of approximate computing via two well-known and broadly-applied recognition algorithms, namely, Canny edge detection [5] and the Scale Invariant Feature Transform (SIFT) [6]. Figure 5 shows the different tasks and their functions and input parameters that are approximated for the two aforementioned algorithms. Both these exemplified applications extract different features from input data for evaluation. We implemented both the applications on computing devices in our testbed using the OpenCV library.

Input data set: We execute our application by using data from the Berkeley image segmentation and benchmark dataset [15]. For offline profiling we used 200 grayscale images from the training data set; for run-time evaluation we used 100 images from the test data set, both available in [15]. Resolution of each image is 481×321 pixels.

Light-weight run-time algorithms: We implement in Android the $\mathrm{Heuristic\ MP-SP}$ algorithm to select the approximation tasks. The algorithms to construct the approximate workflow need to be of low complexity because the gain in reduction in makespan obtained from approximate computing should not be eclipsed by the execution time of algorithms to select the approximate workflow at run-time, which would result in a paradox.

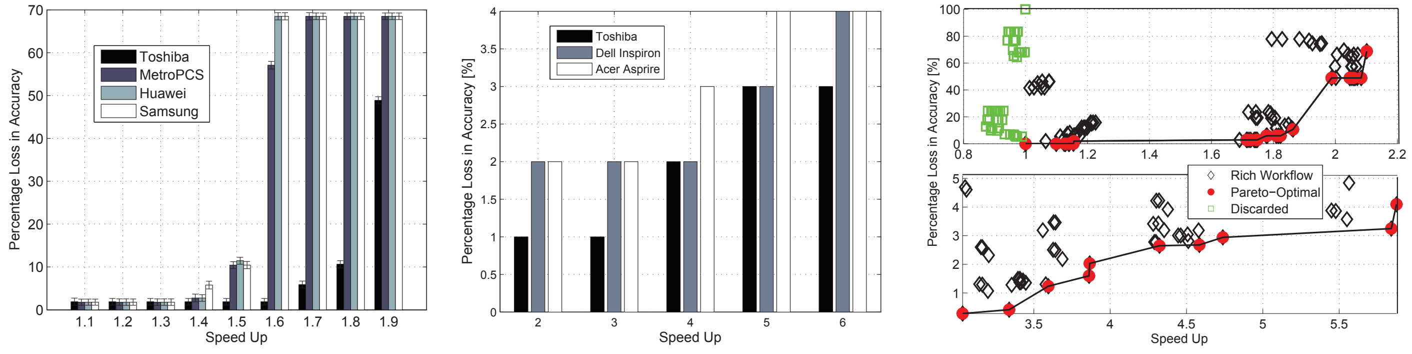
Offline-profiling: Our framework performs offline profiling (as explained in Algorithm 1) of an application. In the profiling phase, we execute the algorithms on the computing devices in our testbed and use input data from the training dataset in [15]. In Table II, we observe how the number of approximate instances in rich workflow and optimized workflow will vary as the acceptable accuracy loss is varied. The number of discarded workflows decreases as the percentage of acceptable accuracy loss is increased; this is because the approximate instances that achieve speed up, although at higher accuracy loss are, also included. Table III quantifies the gain of applying approximation transformations to various tasks and parameters of the aforementioned applications.

Performance of approximate vs. exact computation: Figs. 6(a) and (b) show the results of percentage loss in accuracy obtained when different levels of speed up are achieved by applying approximation transformation to the application. In Fig. 6(a), we see that for Toshiba we achieve a speed up of 1.5 for 5% accuracy loss while for the other devices we get around 10% of accuracy loss. The speed up of Toshiba continues up to 1.9 but it saturates at 1.7 as for the other devices. Similarly, in Fig. 6(b) we see that the speed up is 5 times when the percentage accuracy loss is 3% for Toshiba. Similar trend is observed for the other devices. We can notice that, although the makespan has decreased with different user-specified accuracy bounds, it does not come at the cost of significant loss in accuracy. From the approximate instances, we can determine the Pareto-optimal instances to reduce the approximable task space, as shown in Fig. 6(c). The red dots in the figure indicate the *Pareto Front* for different applications.

Performance of our online algorithm: We compare the performance of our algorithm $\mathrm{Heuristic\ MP-SP}$, which is required to construct an approximate workflow given the run-time application deadline and accuracy loss. We assume the value of $count$, i.e., the number of shortest paths considered in Algorithm 3, to be 3. We compare the performance of our solution, "Optimized-WF Probabilistic," against a deterministic technique, where the delay value of an edge, $d(e)$ in Algorithm 3, is substituted with the mean delay (i.e., the average of delays obtained from different trials executed during the offline phase). We call this approach "Optimized-WF Average." We also compare the performance when Algorithm 3 is applied on a rich workflow instead of the optimized workflow. We call this approach "Rich-WF Probabilistic." In Fig. 7(a), we see that all the techniques are able to give the output within the requested deadline. However, the difference in performance of the techniques is evident in Fig. 7(b), where we see that our Optimized-WF Probabilistic meets the percentage accuracy loss as estimated by the non-linear model (shown in dotted red line). Conversely, for the other two techniques a much higher accuracy loss is incurred in comparison to the expected one. The expected accuracy loss is estimated by the non-linear model discussed in Sect. IV. This is because Rich-WF Probabilistic considers all the approximate instances to select the approximate workflow and misses selection of Pareto-optimal instances, which have slightly higher makespan but incur lower accuracy loss.

Overhead of online algorithm: The overhead of online Algorithm 3 to select approximate workflow at run-time is of the order of 6 ms. This is much lower than the reduction in gain in makespan achieved by approximate computing, which is in the order of hundreds of milliseconds to seconds, as depicted in Fig. 6(a). Hence, our online approximation algorithm incurs a very small penalty, i.e., its overhead is almost negligible compared against the substantial performance benefits it brings in terms of speed up.

Applicability of approximate computing: In Fig. 8(a), we plot different regions of applicability of approximate

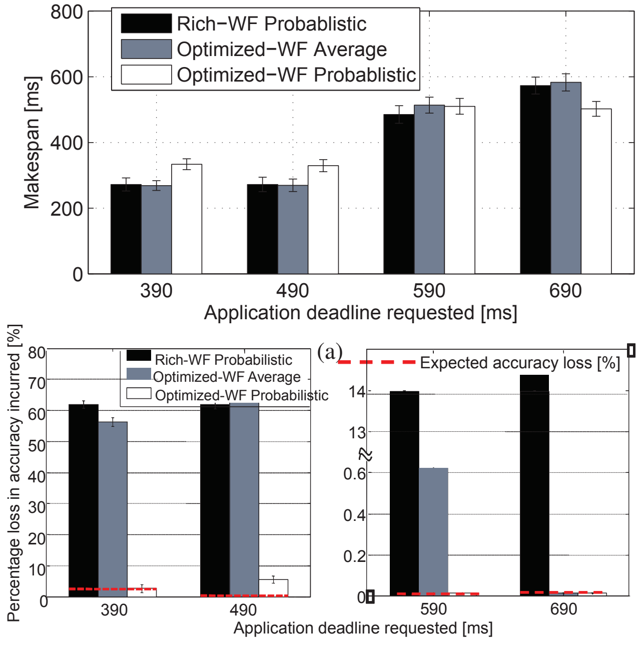


(a) (b) (c)

Fig. 6: **Experiments.** Percentage loss in accuracy versus speed-ups achieved by applying approximate computing techniques on (a) Canny edge detection and (b) SIFT algorithm; (c) (Top) Pareto-optimal instances for Canny edge detection algorithm, (Bottom) Pareto-optimal instances for SIFT algorithm.

computing. If the user cannot tolerate any accuracy loss, e.g., face recognition application to unlock a device or a financial website, then the user is ready to wait for a longer duration and utilize higher resources without any sacrifice of quality, here, exact computing can be applied (seen in the rightmost pink region). Conversely, in interactive applications such as gaming or object recognition, user expect quick response and accuracy loss can be incurred without any perceivable degradation of QoS to the user. Hence, they are good candidates for approximate computing (seen in the leftmost blue region). The situations where the mobile device is limited by battery or does not have enough CPU cycles to give a crisp response to the user, approximate computation is beneficial. Response from cloud applications depend on the network latency; hence, in situations with high cloud latency or with intermittent network connectivity, approximate computing can be applied to give low response time (seen in the middle green region).

Performance of approximate computing in uncertain mobile environment: Our goal is to study the benefits of approximate computing in a Mobile Device Cloud (MDC) where a resource-constrained mobile device offloads its tasks to nearby devices. Uncertainty in a MDC may arise due to *device mobility*, which determines the availability duration of devices, and *network connectivity*, which determines the communication cost of offloading tasks to nearby devices. In our experiments the communication between devices in a MDC is achieved using the AllJoyn framework [16], an open-source, platform-independent software system that provides an environment for distributed applications running across different classes of devices. An AllJoyn thin app is designed for energy-, memory-, and CPU-constrained devices and has a very small memory footprint. Figure 9 shows the architecture for our testbed, which is based on our work in [17]. The service requester device contains the resource task mapper, which is responsible to allocate task to different service providers. We assume a fair, simple, and robust round-robin-based technique to distribute tasks in an MDC.



(a)

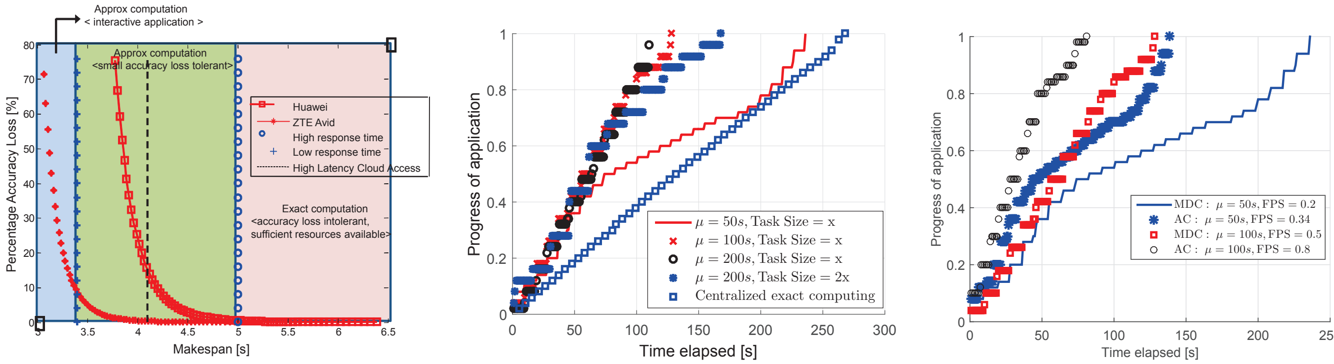
(b)

Fig. 7: **Experiments.** Comparison of probabilistic and deterministic framework in terms of (a) makespan and (b) accuracy loss incurred.

We model the mobility patterns of devices in the proximity as a normal distribution with mean availability duration of devices varying with $\mu = \{5, 100, 200\}$ s and σ=5 s. Our first result shows the gain obtained by the execution in a MDC in comparison to centralized computation. We implement the Canny edge-detection application on devices via exact computation. In Fig. 8(b), we plot the time taken to execute an application as the mean availability duration of the devices in the MDC is varied. Interestingly, as the arrival duration of devices increases, the rate at which tasks are completed increases. Also, in spite of the offloading cost, MDCs finish the execution faster than centralized exact computing. Next, we compare the performance of exact versus approximate execution in a MDC. In Fig. 8(c), we see that by applying approximation we are able to achieve a much higher Frame Per Second (FPS) rate. This is beneficial in case of interactive applications as they are time-critical.

VI. Conclusion

We considered the new paradigm of approximate computing to exploit the untapped potential of mobile distributed



(a) (b) (c)

Fig. 8: **Experiments.** (a) Scenarios where approximate computing can be beneficial in comparison to local exact computing and exact computing in the Cloud. The type of computation depends on several factors such as type of application (interactive or non-interactive), accuracy requirements of the application, resources available, and network latency; (b) Comparison of performance of exact computing in a centralized implementation vs. in a mobile device cloud in the presence of uncertainty in i) device availability due to network disconnections and device mobility and ii) variable task sizes; (c) Comparison of performance of approximate computing vs. exact computing in a mobile device cloud in the presence device mobility.

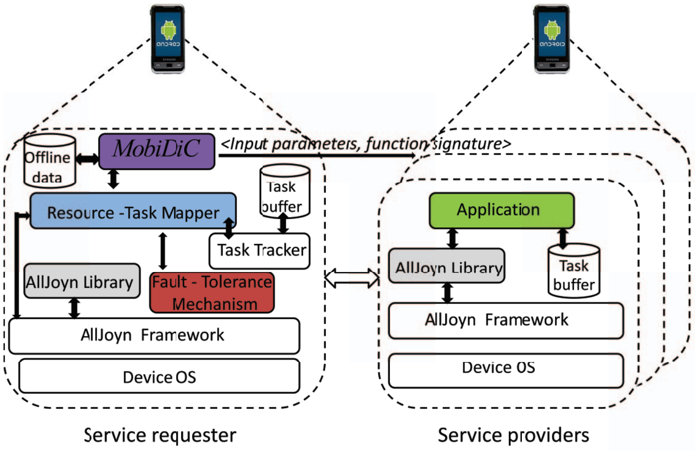


Fig. 9: Testbed to study the gain in performance of approximate computing over exact computing in the presence of uncertainty experienced in a mobile environment.

computing and to enable real-time, pervasive applications in a resource-constrained mobile device cloud. We introduced a Mobile Distributed Computing framework, $MobiDiC$, that determines offline the approximable tasks in an application via a powerful workflow representation scheme. We validated the effectiveness of the proposed approach through extensive simulations and testbed experiments taking as motivating example two different algorithms for interactive perceptive object recognition, and observed that on our testbed their approximate implementations perform better than their exact counterpart.

Acknowledgment: This work was supported by the ONR-Young Investigator Program (YIP) Grant No. 11028418.

References

[1] "Augmented Reality," http://cacm.acm.org/magazines/2014/9/177938-augmented-reality/fulltext.

[2] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud," in *Proc. of European Conference on Computer Systems (EuroSys)*, Salzburg, Austria, April 2011.

[3] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling Interactive Perception Applications on Mobile Devices," in *Proc. of Intl. Conference on Mobile Systems, Applications, and Services (MobiSys)*, Bethesda, MD, Jun. 2011.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proc. of Intl. Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, Jun. 2010.

[5] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986.

[6] D. G. Lowe, "Distinctive Image Features from Scale-invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[7] J. Hwang and P. Aravamudham, "Middleware Services for P2P Computing in Wireless Grid Networks," *IEEE Internet Computing*, vol. 8, no. 4, pp. 40–46, 2004.

[8] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices," in *Proc. of Intl. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Hilton Head Island, SC, June 2012.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *ACM SIGPLAN Notices*, vol. 46, no. 6, 2011, pp. 164–174.

[10] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing Performance vs. Accuracy Trade-offs with Loop Perforation," in *Proc. of European Conference on Foundations of Software Engineering (FSE)*, Szeged, Hungary, Sept 2011.

[11] W. Baek and T. M. Chilimbi, "Green: A Framework for Supporting Energy-conscious Programming using Controlled Approximation," in *ACM Sigplan Notices*, vol. 45, no. 6, 2010, pp. 198–209.

[12] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger, "Eon: A Language and Runtime System for Perpetual Systems," in *Proc. of Intl. Conference on Embedded Networked Sensor Systems (SenSys)*, Sydney, Australia, Nov 2007.

[13] L. J. Van Vliet, I. T. Young, and P. W. Verbeek, "Recursive Gaussian Derivative Filters," in *Proc. of Intl. Conference on Pattern Recognition (ICPR)*, Brisbane, Qld, August 1998.

[14] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate Signal Processing," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 15, no. 1-2, pp. 177–200, 1997.

[15] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," in *Proc. of Intl. Conference on Computer Vision (ICCV)*, Vancouver, BC, July 2001.

[16] "AllJoyn," https://allseenalliance.org/.

[17] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili, "Uncertainty-aware Autonomic Resource Provisioning for Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, pp. 2362–2372, 2015.