Collaborative Computing On-Demand: Harnessing Mobile Devices in Executing On-the-Fly Jobs

Thomas Langford
High Point University
High Point, NC 27262

Qijun Gu
Texas State University
San Marcos, TX 78666

Agustin Rivera-Longoria
Texas State University
San Marcos, TX 78666

Mina Guirguis
Texas State University
San Marcos, TX 78666

***Abstract*—Systems employing mobile devices (e.g., sensors, smart phones, robots) are emerging with growing capabilities in performing a wide variety of tasks. Due to their abundance and wide deployments, they are posed to play a dominant role in providing a rich mobile computing platform for various jobs, especially for new ones that are created on-the-fly. Realizing this platform is challenging since it is hard to predict the exact equipment present in an environment, what types of information need to be communicated to the devices to execute their tasks, and how to reprogram these devices. This work proposes a new on-demand collaborative computing framework that maps a new job as a set of tasks onto the mobile devices for execution. The mapping is done in a manner that takes into account the capabilities of the devices, the dependency between the tasks, the adjacency of the devices, and the requirements of the requested new job. Our proposed framework is implemented as a test-bed in our Mobile Cyber-Physical Systems lab with MICAz sensors and iRobot Create robots.**

I. Introduction

With recent technological advances in embedded systems and wireless communication, mobile devices (e.g., sensors, robots, cell phones, and other various kinds of networked devices) are emerging as powerful devices that can sense, compute, and communicate. It has been evident that the collection of mobile devices can provide a new mobile cloud computing environment where computation can be split and performed on different mobile devices in a coordinated manner [1]–[3]. This new computing approach is posed to revolutionize the way we utilize mobile devices in providing new services and supporting emerging applications in the near future.

Harnessing the capabilities of this collection of mobile devices requires new frameworks with novel fundamental concepts that can take advantage of their diversity. We envision an on-demand computing environment in which new applications/services can be executed on a set of mobile devices – with each device executing a specific subset of the tasks – and the partial results combined to form the ultimate results. Nevertheless, there are many challenges that need to be tackled in order to realize such a rich collaborative computing environment. (a) The available devices may not be exactly known prior to the deployment of new applications, neither are their resources nor their capabilities. For example, we may not know what types of sensors are available for a monitoring application. This requires programming the devices on-the-fly after discovering them. (b) While mobile devices can run specific tasks, they typically do not know how to execute tasks *for a specific new application*. For example, a temperature sensor may not know at what frequency and for how long it should collect temperature for an emerging application. This requires programming the devices in the context of the application. (c) The available resources and the cost of communication between devices may vary a lot based on the nature of the devices (e.g., processor type, available storage, communication methods). This requires methods to find optimized allocations of tasks to devices.

To address these challenges, we present an on-demand collaborative computing framework that is capable of discovering the capabilities of mobile devices present in an environment, mapping a job as task groups to the mobile devices. The mapping of the job components to the devices is achieved in a manner that optimizes different metrics under various constraints (e.g., time to obtain results, limits on data storage, overall communication overhead, power consumption). We believe that our proposed framework is the first to harness the capabilities of a set of mobile devices for executing on-demand jobs while taking into account the dependencies between the tasks, the adjacency of the devices and the degree of replication for certain tasks. The framework is applicable in many emergency response situations where the emergency jobs can be abstracted as workflow models and then mapped for execution on the available mobile devices.

This paper makes the following contributions: (1) Unlike traditional code partitioning methods that simply divide programs into modules, we model a job as a workflow abstraction that can be mapped for execution on a collection of mobile devices with the execution of certain tasks pinned to some devices. The workflow model captures the inherent dependencies between tasks along with various replication strategies for common tasks. (2) We formulate an optimization problem for task assignment in which the mapping of a job to devices optimizes different metrics based on the needs of the applications. (3) We develop a programming framework that creates programs to be executed on the devices according to the task assignment and the types of devices. (4) We evaluate our proposed framework in our Mobile Cyber-Physical Systems Lab using realistic test-beds made of MICAz sensors and iRobot Create robots.

In the following, we first overview our proposed framework (Section III), and then discuss in details the abstraction model of a job (Section IV), the job mapping (Section V), the

framework for building diverse device programs (Section VI), and the task dissemination (Section VII). Finally, we present the implementation and evaluation of the proposed framework in Section VIII and conclude in Section IX.

II. Related Work

Recently, there has been a growing interest in utilizing mobile devices in building a mobile computing system. The work in this paper relates to two main areas of research: partitioning and offloading computation in mobile cloud computing, and building and programming mobile cyber-physical systems.

A. Mobile Cloud Computing

In mobile cloud computing, partitioning programs has been proposed as a means to offload the execution of some tasks to the cloud [3]–[11]. A typical approach, such as [4], [6], [7], [9], is to offload mobile code to the cloud in order to save resource on the mobile devices. The problem is modeled as an optimization one with the goal to maximize the energy saved, subject to meeting certain deadlines set by the program. A few works [3], [10], [12] have looked into the possibility of utilizing peer mobile devices to provide computing service. In [12], a probabilistic framework was proposed in which a program can be partitioned into modules that are mapped for execution on a set of mobile devices to ensure dealing with uncooperative and malicious devices without the need for connections to the cloud. In [3], [10], different middleware frameworks were proposed to support offloading mobile code to other mobile devices. The middleware frameworks coordinate offloading and execution among mobile devices.

One common objective of these existing works is to allow a mobile device to utilize other computing resources to accomplish its own work. They target addressing the problems of saving computation for the offloading devices that offload computation to other devices. The offloading devices also need to coordinate their computation with other devices. In contrast, our work investigates how to map a job to a set of available devices so that the devices can perform a portion of the job with their own resources and then collectively complete the job. The job itself is not initiated or offloaded by any device, and the coordination among the participating devices is determined by the job components as they execute.

B. Mobile Cyber-Physical Systems

There have been some efforts in building and programming mobile cyber-physical systems using the combination of robots and sensors. In [13], the authors developed a test-bed for evaluating mobile cyber-physical systems using a number of autonomous robots and sensors. The goal was to provide a scheme to drive system-wide assessment that cannot be simply carried out with simulation experiments. In their work, the experiments were predetermined so there was no need to discover, mapping, and deploy code on the devices. In our framework, we do not limit the system to specific devices, but rather it seeks to discover the devices and their capabilities and to come up with a mapping of various modules onto those devices. In [14], a programming framework was developed to deploy a network of sensors that are linked to actuators in a customizable manner. The framework uses a logic-based programming paradigm that enables the dynamic progammability and configuration of sensor-actuator interactions in wireless sensor networks. However, the programming approach is suitable to computations that can be expressed logically. The complexity of many applications is beyond logic expression. Our framework models a job as a workflow, which is not limited to logic expression, and can support general applications.

III. System Overview

In general, an on-demand computing system consists of a controller and a number of participating mobile devices. The controller is responsible for discovering the participating devices, and disseminating executables to the devices to accomplish the requested jobs. The participating devices are assumed to be not known prior to the system and thus do not carry any executables to run on any desired job. Rather, they participate either on a volunteer basis or an incentive one [3]. Incentives for registering mobile devices for executing on-demand tasks include, but are not limited to, (i) they will have higher access privileges to information that they otherwise cannot access; (ii) they will earn credits for completing the on-demand tasks that can be used to pay their bills; and (iii) they can initiate their own jobs and request the system to utilize other participating devices in carrying out the tasks.

The system has a set of jobs that could be created in advance as a part of an emergency plan. Alternatively, they may be created on demand for occasional incidents. However, a job is not an implemented program, but rather an abstraction of the operations to be performed. A job is designed as a workflow, which consists of a set of tasks and a set of edges. The tasks encapsulate a set of specific operations, and the edges represent the data exchange between tasks. The system maintains a pool of pre-coded task modules for pre-planned and potential jobs. Each task module is programmed for a set of platforms so that they can be built for and executed on various types of mobile devices later.

When a job is requested, the system dispatches the controller (for example, a robot) to the scene in which the job is to be executed. The controller carries the abstraction of the job and the pre-coded task modules. The controller finds and locates the participating devices and obtains certain information from them, such as their available computing resource, their communication capability, their platform type, their hardware functionality, and so on. After the controller gets a list of the participating devices and their information, the controller maps the abstraction of the job to the identified devices according to their capabilities. Then, the controller maps the job as several groups of tasks to identified devices. The controller builds on-demand adaptive programs (ODAPs) using the pre-coded task modules according to the task assignment. Then, the controller disseminates the ODAPs to the devices. Finally, the devices execute the ODAPs and accomplish the job.

IV. An On-Demand Computing Model

In this section, we discuss the modeling of a computing job and the key features of job components that make the modeling unique and suitable for deploying a job to a collection of devices.

A. The Abstraction Model of a Job

A computing *job* is defined as a workflow that is composed of nodes and edges. A node represents a solitary *task*, which is a closed set of operations that must be performed by one device. To meet the requirements of the job, each task specifies a set of computing constraints. An *edge* represents passing the results form an earlier task to a later one. Each task has ingress ports and egress ports that act as its interfaces. When a task is executed, the result of the task is passed along the edge from its egress port to the ingress port of the later task. Then, the later task is executed to process the input data.

An example of the workflow of a computing job is illustrated in Figure 1. The job is to locate a person in a shopping mall. To accomplish this job with using only the security cameras deployed in the mall may not be sufficient, because they cannot look into hidden spots and can only provide image information. Also, the job (locating a person) itself may not be a part of the work of the mall's monitoring system. Hence, the proposed on-demand computing framework will very match the needs of this job, since it can disseminate the tasks to the proper mobile devices in the mall and ask them to help locating the person.

The job is designed to have 6 tasks: capturing images of people in the mall (T1), recording voices (T2), storing the captured images (T3), recognizing features of the target person (T4), sanitizing stored data to remove sensitive personal and business information (T5), and identifying the possible locations of the target person (T6). The edges of the workflow indicate how the captured image and voice data are processed by the tasks.

Besides utilizing the mall's monitoring system, the job adds the voice data and asks the shoppers to use their carry-on devices to help locate the person. T1 and T2 are the tasks for multiple mobile devices with cameras or microphones, such as smart phones of shoppers. Hence, they can be replicated to multiple shoppers and provide better coverage. The other tasks mainly process data. They can be disseminated to the computers of the mall or the laptops and pads of the shoppers. Both image and voice data will be used by T3 to recognize the features of the target person.

B. Properties of Job Components

In contrast to a typical computation abstraction in mobile cloud, the workflow model of a computing job in our work considers the execution of multiple devices. For example, the same task can be assigned to multiple devices, and different tasks can be executed in different devices simultaneously. These considerations are included as properties of job components and allow the framework to maximally utilize available devices to complete a job.

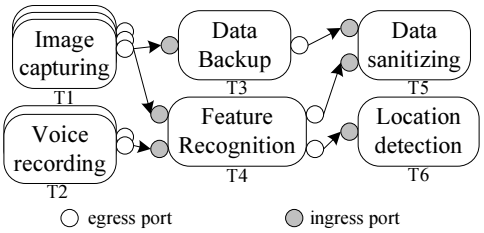

Fig. 1. Example of a Workflow

1) Task Properties: A task has three properties as its signature. We can follow the three properties to build different ODAPs of the same task for different devices.

The essential property of a task is its *functionality*, which defines the operations that the task should perform to accomplish the job. But, this property does not specify the exact coding of the task. Instead, the task will be programmed according to the actual mobile devices. In Figure 1, T1 can be performed by a mall's security camera or a shopper's smart phone. The actual programs for the two devices are obviously different.

Another property is its *replication* – whether a task can be replicated in a workflow or not. A replicable task is used when multiple devices can be asked to work on the same task. In Figure 1, T1 and T2 can be replicated to multiple capable devices so that they can obtain image and voice data for the job. But, in this example, the other tasks are not replicated because they need to aggregate and process the obtained data in a centralized manner.

The last property is its *communication interface* – the set of ingress and egress ports of a task along with the data format passed through the ports. In Figure 1, T4 has two ingress ports. One accepts voice data and the other accepts image data. It has two egress ports. One outputs the recognized image features to T5 to sanitize backuped image data and the other outputs the aggregation of the recognized image and voice features to T6 to detect locations.

2) Edge Properties: The property of an edge specifies how to pass data between two ports. It defines the format of data passed along the edge, i.e. the number of data items, and the type and size of each data item. Correspondingly, the egress and the ingress ports of one edge shall have the same data format. Because a port can only send or receive data in one format, all edges that start from the same egress port or end at the same ingress port have the same data format. In Figure 1, the edges between the egress port of T1 and the ingress ports of T3 and T4 have the same data format, even though they pass data among different pairs of tasks.

V. Job Mapping and Task Assignment

Once the workflow of a job is created, the job is divided to several groups of tasks. Because it is not known in advance what and how many devices can be utilized, the task groups are created on demand and adaptive as well. In the following, we first explain the idea of task groups, and then formally model the job mapping problem.

A. Task Groups

1) Grouping Tasks under Resource Constraints: Although a job is made of individual tasks, a capable device can execute multiple tasks. Therefore, the objective of an effective job mapping method is to group tasks according to the capabilities of the available devices.

Let T_i be the i-th task, and D_p be the p-th device. Let r_i be the resource requested by T_i, and R_p be the resource available at D_p. Let G_p be the group of tasks assigned to D_p.

The mapping shall group adjacent tasks for a device so that the device can provide the requested resources to complete the assigned tasks and achieve the desired performance. The resource constraints are expressed in Formula (1), where $RS()$ is a resource function. For example, $RS()$ is a summation function if the requested resource is storage.

$$\forall p, RS(\{r_i : T_i \in G_p\}) \leq R_p \tag{1}$$

2) External Edges and Overhead Constraints: After mapping, the tasks are assigned to different devices. The edges among the tasks inside the same device become the *internal edges*, while the edges connecting the tasks in different devices become the *external edges*. Passing data along the external edges, however, incurs extra communication overhead due to the delay and the bandwidth consumed for passing data. Therefore, job mapping needs to consider such performance overhead as well.

Let E_i be the i-th external edge, and o_i be the communication overhead on E_i. For example, o_i is the communication delay over E_i. Let O_i be the overhead tolerance on E_i. The tolerance reflects the maximum overhead that will not affect the job. Thus, the overhead constraints can be expressed in Formula (2).

$$\forall i, o_i \leq O_i \tag{2}$$

3) Special COM Task: Passing data along the external edges requires communication between devices. Hence, we create a special communication task (COM) to serve this purpose. For each port connected with an external edge, a COM task is inserted between the port and the external edge. Note that a COM task may be created alone for a device if the device is only needed to forward data.

A COM task takes four arguments: the source egress port, the destination ingress port, the size of data, and the data to be transmitted. Since devices use different communication and networking protocols, COM tasks will communicate according to the protocols present.

4) Job Mapping Example: Figure 2 illustrates an example of job mapping for the workflow presented in Figure 1. Six devices are identified and assigned proper tasks according to their capabilities so that they can perform the assigned tasks as well as satisfy both resource and overhead constraints. For example, T3 and T5 are assigned to a computer D4 with a sufficient storage. D5 and D6 are shoppers' carry-on pads. They are then utilized for T4 and T6.

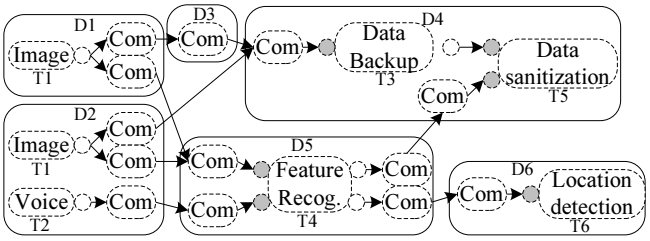

Fig. 2. Example of Job Mapping

T1 and T2 are replicable tasks. Assume multiple devices (D1 and D2) are identified for the two tasks. D1 is a security camera that can only capture images, while D2 is a smart phone that can capture both images and voices. Hence, T1 is assigned to D1 and T1 and T2 are assigned to D2.

Several COM tasks are created for external edges. But, D3 is a special case because it is needed so that D1 can send data to D4. Hence, D3 only runs the COM task to forward data between D1 and D4, and does not perform any tasks in the original workflow.

B. Problem Formulation of Job Mapping

Given a list of available devices, a job may be mapped in different ways. Therefore, it is necessary to find the best mapping that achieves a certain optimal objective while satisfying both resource and overhead constraints.

Various optimal objectives can be defined under the resource and performance constraints. For example, we can set the optimal objective to maximizing the number of utilized devices so that the job can be supported by as many devices as possible and no device will be overloaded to affect their regular work.

To formulate the job mapping problem, we first define the following binary integers, using the same notations in Formulas (1) and (2).

- Let $b_{p,q} = 1$ represent that devices D_p and D_q are neighbors, and 0 otherwise.
- Let $e_{i,j} = 1$ represent that tasks T_i and T_j are connected by edges, and 0 otherwise.
- Let $x_{i,j} = 1$ represent that the edges between T_i and T_j are external, and 0 otherwise.
- Let $s_{i,p} = 1$ represent that task T_i is assigned to the device D_p.

Assume a job is made of N tasks and M edges, and L devices are available for this job. Among the tasks, $S_{\bar{R}}$ is the subset of all non-replicable tasks and S_R is the subset of all the other replicable tasks. The job mapping problem is then formulated as below.

Optimization:

$$argmax_{\{s_{i,p}\}} \sum_p (\max_i s_{i,p}) \tag{3}$$

Constraints:

$$\textstyle\sum_p s_{i,p} = 1, \forall T_i \in S_{\bar{R}} \tag{4}$$

$$\textstyle\sum_p s_{i,p} \geq 1, \forall T_i \in S_R \tag{5}$$

$$\textstyle\sum_i s_{i,p} r_i \leq R_p, \forall D_p \tag{6}$$

$$x_{i,j}o_{i,j} \leq O_{i,j}, \forall T_i \neq T_j \quad (7)$$

$$s_{i,p}s_{j,q}e_{i,j} = x_{i,j}, \forall T_i \neq T_j, D_p \neq D_q \quad (8)$$

$$s_{i,p}s_{j,p}x_{i,j} = 0, \forall T_i \neq T_j, D_p \quad (9)$$

$$s_{i,p}s_{j,q}e_{i,j} \leq b_{p,q}, \forall T_i \neq T_j, D_p \neq D_q \quad (10)$$

$$\forall s_{i,p}, x_{i,j} \in \{0, 1\} \quad (11)$$

The given parameters to the optimization problem include $e_{i,j}$, $S_{\bar{R}}$, S_R, r_i, R_p, $o_{i,j}$ and $O_{i,j}$. The optimization in Formula (3) is to find the best task assignment $s_{i,j}$ that maximizes the number of the selected devices that the tasks are assigned to.

Constraint (4) states that a non-replicable task shall be assigned to only one device. Constraint (5) states that a replicable task shall be assigned to at least one device. Constraint (6) formulates the resource constraints in Formula (1). Constraint (7) formulates the overhead constraints for external edges in Formula (2). Constraint (8) states that if two adjacent tasks are assigned to two different devices, the edge of the two tasks shall be an external one. Constraint (9) states that if two adjacent tasks are assigned to the same devices, the edge of the two tasks shall be an internal one. Constraint (10) states that if two adjacent tasks are assigned to two different devices, the two devices shall be neighbors. Constraint (11) states that $s_{i,p}$ and $x_{i,j}$ of the optimization are either 0 or 1.

The formulated job mapping problem is essentially an integer programming (IP) problem. It can be solved or approximate solutions can be found with IP solvers [15]. However, studying the best algorithms for optimal job mapping is not the goal of this paper. We will address the algorithms for this problem in our future work.

VI. Framework of On-Demand Adaptive Programs

As a job is mapped to multiple groups of tasks and then disseminated to different devices, the actual programs running inside the devices vary and are created on demand as well. It is thus necessary to have a software framework to facilitate making ODAPs in the controller for the selected devices.

The proposed ODAP software framework is made of three parts. One part is the *ODAP template* which includes the basic routines to initialize the device, start the execution of ODAP, and accept new ODAPs. Another part is the *ODAP task modules*, which execute the assigned tasks and interact with other tasks or devices. The template has a programming interface to add different task modules. The last part is the *ODAP workflow*, which captures the subgraph of the job's workflow. Each subgraph includes the groups of the tasks and their edges assigned to a device.

For each specific type of device, the template and the task modules are coded in advance. When deployed, the controller generates the ODAP workflows based on the job mapping, and then uses the ODAP framework of a specific device to plug the device-specific modules into the device-specific template to create an actual executable ODAP. The ODAP is then disseminated to the device.

The template, the task modules, and the workflow are programmed in XML and high-level programming languages

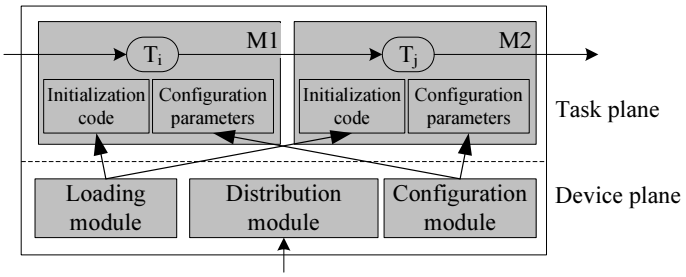

Fig. 3. ODAP Template

for each device so that the whole ODAP can be best optimized monolithically to produce the final executable for efficient execution. This approach also provides flexibility in code development and management.

Without loss of generality, we will use MICAz and TinyOS [16] as the platform and the programming language to show the coding examples in the discussion.

A. ODAP Template

The ODAP template is the most basic structure of an ODAP. It structures an ODAP with a device plane and a task plane, as illustrated in Figure 3. The template implements the mechanism of adding the modules (grey boxes) in the device plane and the task plane to an ODAP and the mechanism for the modules to interact. The template itself is not task-dependent and does not implement any functionality of the modules.

The device plane includes a loading module, a distribution module, and a configuration module. The loading module starts the ODAP and then initializes the task modules. The distribution module broadcasts the capability of the device, and takes new ODAPs from the external controller in order to execute new tasks. The configuration module allows an external controller to remotely configure the parameters used by the task modules. The three modules of the device plane are not task-dependent and should be able to run with no task modules added.

The task plane of the template is a container to include the assigned task modules. It tags the proper locations where the task modules are filled in so that the ODAP can execute the assigned tasks. The task plane specifies four tags for adding task modules to the ODAP. One is the module declaration tag "`<Components>`", which declares and includes the needed task modules in the ODAP. The second is the initialization tag "`<Inits>`" that allows the loading module to initialize the task modules. The third is the task tag "`<Tasks>`" that is to include the actual workflow of the assigned tasks. The last is the configuration tag "`<Configs>`" that allows the configuration module to configure the task modules.

An example of the ODAP template for TinyOS is illustrated in Figure 4. Although the example template is programmed and structured in XML, it follows the programming specification of TinyOS. A portion of the TinyOS code is substituted with XML notations in the template so that the controller

```
<Template>
  <TemplateAppC>
    configuration <ODAppC/> {}
    implementation {
      components <LoaderC/>, MainC,
        DelugeC, ConfigC, ...;
      <Components/>
      <Wiring/>
    }
  </TemplateAppC>
  <TemplateC>
    module <LoaderC/> {
      uses {
        <Interfaces/>
      }
    }
    implementation {
      event void Boot.booted() {
        <Inits/>
        ...;
      }
      <Tasks/>
      <Configs/>
    }
  </TemplateC>
</Template>
```

Fig. 4. Example of an ODAP Template XML for TinyOS

```
<Mod>
  <Component>AverageC</Component>
  ...
  <Init>
    call Average.init(<C id="0"/>);
  </Init>
  <Config id="0">
    call Average.config(cfg, len);
  </Config>
  <Ingress id="0">
    <Parameters>...</Parameters>
    call Average.average(<P id="0"/>);
    <Com>
      <Parameters>...</Parameters>
      call Average.average(<P id="1"/>);
    </Com>
  </Ingress>
  <Egress id="0">
    <ReturnVals>...</ReturnVals>
    event void Average.avgDone(uint16_t avg) {
      <Return>
        <R id="0">avg</R>
      </Return>
    }
    <Com>
      ...
    </Com>
  </Egress>
</Mod>
```

Fig. 5. Example of an ODAP Task Module XML for TinyOS

can generate different ODAPs for different jobs and devices. The template itself will be transformed by the controller to a TinyOS coding file.

In the device plane, the loading module is "LoaderC", the distribution module is "DelugeC" [17], and the configuration module is "ConfigC". The three modules are made in advance. The four key tags in the task plane are highlighted and later will be substituted with task code. All other tags in the template will be substituted as well with actual TinyOS code according to the task assignment.

B. Task Module

A task module contains the actual code for executing a specific task. As discussed in Section IV-A, a task module includes both the ports and the functionality of a task. Following the conventional modular design approach, the ports are the interfaces of the task module, and the functionality is encapsulated inside the module.

Besides the task code, two auxiliary routines are added to the task module. The initialization routine is to initialize the computing environment for the task. The configuration routine is to configure the parameters used by the task. For example, a task averages the received data items periodically. The task needs a counter and a period. The counter is a variable that needs to be set to zero by the initialization routine when the task is started. But, the period is a parameter that needs to be configured by the configuration routine for a proper average.

Thereby, a task module includes the task code, the ports, the two auxiliary routines, and the configuration parameters. To manage the task code, a task module is made of an task XML file and the actual coding files. The task XML file contains the information for the ports, the auxiliary routines and the configuration parameters. The task XML file is used by the controller to plug the task into the ODAP template and only includes the code necessary to substitute the tags defined in the template. The majority of task code is kept in the coding files.

An example of the task XML file is illustrated in Figure 5. The two auxiliary routines are tagged with "`<Init>`" and "`<Config>`". The code inside the two tags will be used to substitute the tags "`<Inits>`" and "`<Configs>`" in the ODAP template. The ingress port of the task is tagged with "`<Ingress>`", and the egress port is tagged by "`<Egress>`". The codes associated with the port tags will be used to substitute the tag "`<Tasks>`" in the template. Note that the code in the ports include two parts. One is for interacting with another regular task module, while the other is for interacting with a COM task.

C. Workflow

The ODAP framework encodes the workflow in XML as well. For each device, the workflow is a graph of the tasks assigned to the devices and the edges connecting the tasks. Thereby, the workflow for each device is generated by the controller based on the results of job mapping. The workflow has two components: tasks tagged by "`<Task>`" and edges tagged by "`<Edge>`". Different from the template and the task module, the workflow is not platform-dependent.

```
<ODApp>
  <Tasks>
    <Task id="0" mod="0">
      <Config id="0" val="3"/>
    </Task>
    ...
  </Tasks>
  <Edges>
    <Edge id="0">
      <Head mod="1" egress="0"/>
      <Tail id="0" mod="0" ingress="0"/>
    </Edge>
    ...
  </Edges>
</ODApp>
```

Fig. 6. Example of an ODAP Workflow XML

An example of the workflow XML file is illustrated in Figure 6. Each task includes the "<Config>" tag that provides the initial values to the configuration parameters. Each edge includes the "<Head>" tag and the "<Tail>" tag that provide the information of the task modules and the corresponding ports connected by the edge.

VII. ODAP Dissemination

Two main components are involved in code distribution. One is the code distribution process. The other is the preparation and organization of task modules.

A. Search and Dissemination Process

The controller interacts with mobile devices through their distribution modules for code dissemination. The process has two phases: search and dissemination. In the first phase, the controller searches for devices and collects the capability information of the available devices. Such information includes the available computing resources (such as storage, energy and CPU speed) and the device information (such as the hardware platform, the software system, the type of sensors and the type of communication). Once the controller gets a list of devices and their capability information, the controller conducts the job mapping. If the controller can successfully map the job to the identified devices, the controller will continue to the next phase. Otherwise, the controller will continue the search until it finds sufficient devices to take the job.

In the next phase, the controller builds ODAPs according to the job mapping. For each device, the controller locates the task modules for the assigned tasks and the specific device type, and then generates the ODAP using the ODAP framework. Then, the controller disseminates the ODAP to the device. Upon receiving the new ODAP, the device reloads or reprograms itself with the ODAP, and then runs the assigned tasks.

B. Task Modules for Devices

Because different devices may be utilized, the controller group possible devices into categories according their hardware and software architectures. For each device category, task modules are made in advance as if the complete job is performed by a single device. A module manifest is also made in advance to include the information of the task modules for all device categories. The controller carries the manifest in the aforementioned search and dissemination process.

```
<ModList>
  <Mod id="0" func="0">
    <Description>Average</Description>
    <FilePath>modules/averageMod.xml</FilePath>
    <Platform dev="MICAz">TinyOS</Platform>
  </Mod>
  <Mod id="2" func="2" com="true">
    ...
  </Mod>
  ...
</ModList>
```

Fig. 7. Example of a Module Manifest XML

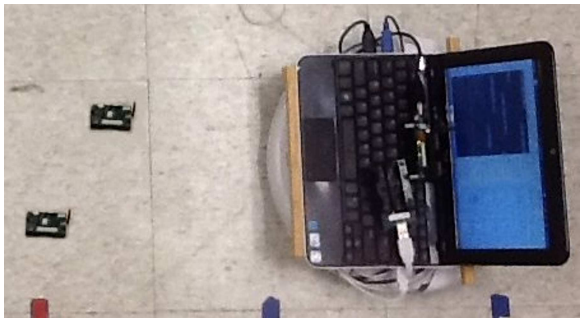

Fig. 8. POC Test-bed

An example of the manifest is illustrated in Figure 7. Each "<Mod>" tag includes (a) a description of the module's functionality, (b) the location of the module's code, and (c) the platform of the device running this module. It also indicates if the module is a COM module.

VIII. Evaluation

We implemented a proof-of-concept (POC) mobile test-bed for evaluating the proposed framework.

A. Implementation

1) Hardware: Figure 8 illustrates the POC test-bed. It has a iRobot Create as the base for mobility. It communicates through a serial port to a Dell Inspiron netbook running Fedora 17. The netbook is also connected to two MICAz motes running TinyOS through another two serial ports. One mote is programmed to listening to the participating motes and the other is to disseminate ODAPs to the selected devices.

The netbook runs the operations. It takes in all of the inputs from the robot and the motes. It directs the movement of the Create. When the Create moves, the listening mote detects other devices. Upon getting a list of devices, the netbook computes the job mapping and generates the ODAPs. Then, the netbook moves to the selected devices and disseminates the ODAPs to them using the disseminating mote.

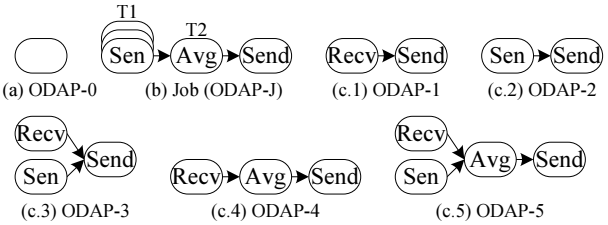

Fig. 9. ODAPs in Experiments

2) Software: The test-bed modified Deluge for disseminating ODAPs. Deluge is the code distribution component in TinyOS. But, it disseminates programs over a multi-hop wireless network to all devices [17]. For the purposes of the on-demand computing system, it is desired that each ODAP is distributed to only one device as opposed to every device in the network. This enables the controller to disseminate different ODAPs to different devices in the network.

B. Experimental Settings

We conducted a variety of experiments with the test-bed. The job of the experiments is to monitor a target. The workflow of the job itself is in Figure 9(b). As discussed in Section IV-A, the workflow was created without considering the actual implementation, but focused on the tasks of the job. In particular, the tasks of the job are monitoring (T1) and processing (T2). In experiments, T1 was defined as sensing, and T2 was defined as averaging the sensing data.

We assume the target is far away. Thereby, a network is needed for delivering data. We also assume we do not know in advance what devices around the target can be used for the job and what devices can be used to make a network to deliver the monitor data back to the monitor center. We do not program the sensors for this job in advance either, since we assume the sensors were deployed before the job occurs.

We deploy several MICAz motes to emulate devices that have different functionality. In experiments, we set up some MICAz motes with sensors and the others without. Because the network is not a part of the job's workflow, the COM tasks will be assigned to some motes to form an on-demand ad hoc network. When the controller roams, it detects the motes and disseminates the job. A video demonstration of the experiments is available at http://www.youtube.com/watch?v=JaoTun-uX7o.

C. Experimental Results

For comparison, we enumerate all possible ODAPs resulting from the job mapping in Figure 9 and include two baseline ODAPs as well. One baseline is the ODAP-0 in 9(a), which has only the device-plane components. As discussed in Section VI-A, ODAP-0 shall run without any tasks. The other baseline is the ODAP-J in 9(b), which is for a hypothetical scenario where one device can accomplish the whole job. Other than the two baseline ODAPs, the other five ODAPs in Figure 9(c.1) through (c.5) are possible ODAPs assigned to motes. ODAP-1 is for the communication motes that only forward data. ODAP-2 is for the motes that sense only. ODAP-3 is for the motes that sense and forward data. ODAP-4 is for the mote that only averages the sensing data. ODAP-5 is for the mote that can sense, average and forward data.

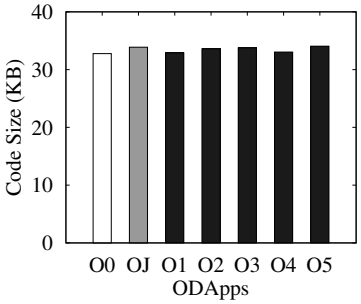

Fig. 10. Code Memory Used by the ODAPs

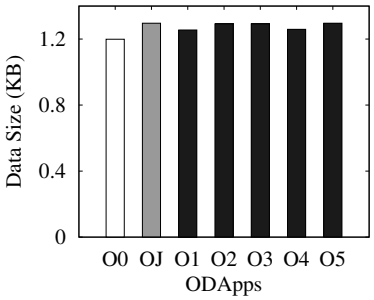

Fig. 11. Data Memory Used by the ODAPs

These ODAPs are disseminated to the selected motes. Instead of asking the motes to route by themselves, these ODAPs are configured with proper next-hop addresses for forwarding data. We measured and compared the computing resources and communication overheads after the ODAPs are deployed in the motes. Note that the process of creating, compiling and disseminating ODAPs is a one-time process in each experiment. The cost and performance of the process do not affect the job, and thus are not included in evaluation.

1) Computational Overhead: These ODAPs were compiled for studying their resource requirements. Figures 10 and 11 show their code size and data size. All the results are compared to the ODAP-0, because it does not carry any task and all ODAPs shall have the code of ODAP-0. The table also shows the increment in code and data due to the assigned tasks. Although the ODAPs were created from a set of code components, the resulting code size and data size are not a simple summation of the code components due to optimization in compilation.

The code components in the experiments include the loading module, the distribution module, the configuration module, the COM modules (*Send* and *Recv*), the sensing module (*Sen*), and the average module (*Avg*). Among these modules, the average module and the configuration module have the minimum amount of code. Meanwhile, the COM modules share the radio and communication code with the distribution

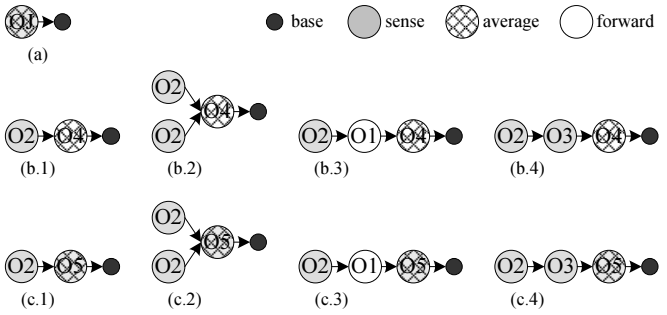

Fig. 12. Sensor Networks with Deployed ODAPs

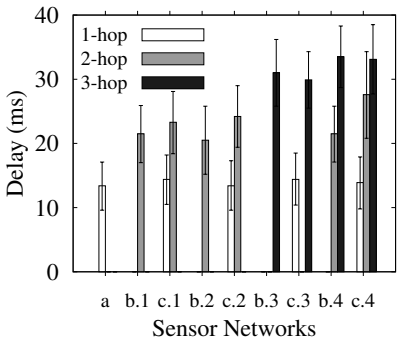

Fig. 13. Delay

module. Therefore, the code and data sizes of ODAP-J, ODAP-2, ODAP-3 and ODAP-5 are similar because they all have the sensing module. In contrast, ODAP-0, ODAP-1 and ODAP-4 have similar but smaller code and data size, because they do not have the sensing module.

2) Communication Overhead: Because the job is split and disseminated to different devices, executing ODAPs incurs communication overhead to the job. To study the communication overhead, the ODAPs in Figure 9 were deployed to sensors to form a variety of experimental sensor networks as illustrated in Figure 12 to accomplish the job. The job itself (in Figure 12(a)) is used as the baseline that simply senses and sends data to a base station. The code deployed to each mote is marked inside its circle. The motes in gray will sense, the mote in lattice will average data, and the motes in white will forward data.

Because more hops are needed to deliver data, extra delays are incurred with the deployed ODAPs. Figure 13 shows the delays in the experimental sensor networks. The delays were measured from when the motes sense to when the base station receives the data. The maximum hop number is three in experiments for the networks of Figure 13(b.3), (b.4), (c.3) and (c.4). Compared with the baseline, the 1-hop delay is around $13.9ms$ and almost same in all networks. The 2-hop networks add an extra delay of about $9.2ms$, and the 3-hop networks add an extra delay of about $18.0ms$. Hence, the additional delay due to multiple hops is proportional to the number of hops, which is reasonable.

IX. Conclusion

This paper presented a new on-demand computing framework that generates mobile programs on-the-fly according to the needs of emerging jobs and the capabilities of the available mobile devices. The paper modeled a job as a workflow abstraction and formulated the problem of job mapping and assignment as a constraint optimization one. The paper provided a new programming framework that manages the code modules for a diverse set of devices and builds mobile programs according to the task assignments and the types of devices. The conducted experiments in our test-bed validated and demonstrated the feasibility of the proposed on-demand computing framework.

Acknowledgment

This research is funded by NSF award #1156712 that is co-funded by the DoD and by NSF award #1149397.

References

[1] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. of ACM Workshop on Mobile Cloud Computing Services*, 2010, pp. 1–6.
[2] M. Satyanarayanan, "Mobile computing: the next decade," in *Proc. of ACM Workshop on Mobile Cloud Computing Services*, 2010.
[3] E. Miluzzo, R. Cáceres, and Y.-F. Chen, "Vision: mClouds - computing on clouds of mobile devices," in *Proc. of the ACM workshop on Mobile cloud computing and services*, 2012, pp. 9–14.
[4] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proc. of Usenix HotOS*, 2009.
[5] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Proc. of ACM/IFIP/USENIX International Conference on Middleware*, 2009, pp. 83–102.
[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
[7] 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 ACM MobiSys*, 2010, pp. 49–62.
[8] B.-G. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proc. of ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, 2010.
[9] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing," *Journal of Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.
[10] H. Eom, P. St Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "SNARF: a social networking-inspired accelerator remoting framework," in *Proc. of the Workshop on Mobile Cloud Computing*, 2012, pp. 29–34.
[11] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proc. of the ACM workshop on Mobile cloud computing and services*, 2012, pp. 21–28.
[12] M. Guirguis, R. Ogden, Z. Song, S. Thapa, and Q. Gu, "Can You Help Me Run These Code Segments on Your Mobile Device?" in *Proc. of IEEE Globecom*, 2011.
[13] C. Fok, A. Petz, D. Stovall, N. Paine, C. Julien, and S. Vishwanath, "Pharos: A Testbed for Mobile Cyber-Physical Systems," *Tech. Report TR-ARiSE, University of Texas at Austin*, 2011.
[14] Y. Wu and A. Rowe, "Logic-Based Programming for Wireless Sensor-Activator Networks," in *Proc. of IEEE/ACM ICCPS*, 2011, pp. 163–173.
[15] R. L. Rardin, *Optimization in Operations Research*. Prentice Hall, 1997.
[16] "TinyOS," http://www.tinyos.net.
[17] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. of ACM SenSys*, 2004, pp. 81–94.