Panorama: A Framework to Support Collaborative Context Monitoring on Co-located Mobile Devices

Khaled Alanezi[1(✉)], Xinyang Zhou[2], Lijun Chen[1,2], and Shivakant Mishra[1]

[1] Department of Computer Science, University of Colorado, Boulder, CO, USA
{Khaled.Alanezi,mishras}@colorado.edu
[2] Interdisciplinary Telecom Program, University of Colorado, Boulder, CO, USA
{Xinyang.Zhou,Lijun.Chen}@colorado.edu

Abstract. A key challenge in wide adoption of sophisticated context-aware applications is the requirement of continuous sensing and context computing. This paper presents Panorama, a middleware that identifies collaboration opportunities to offload context computing tasks to nearby mobile devices as well as cloudlets/cloud. At the heart of Panorama is a multi-objective optimizer that takes into account different constraints such as access cost, computation capability, access latency, energy consumption and data privacy, and efficiently computes a collaboration plan optimized simultaneously for different objectives such as minimizing cost, energy and/or execution time. Panorama provides support for discovering nearby devices and cloudlets/cloud, computing an optimal collaboration plan, distributing computation to participating devices, and getting the results back. The paper provides an extensive evaluation of Panorama via two representative context monitoring applications over a set of Android devices and a cloudlet/cloud under different constraints.

Keywords: Collaborative computing · Pervasive computing · Multi-objective optimization

1 Introduction

In the field of context-aware computing, a wealth of clever mobile applications that monitor user environment to detect and react to events of special interest have recently been proposed; see, e.g., [11,19,20]. However, a major obstacle towards wide adoption of context-aware applications is the requirement of continuous context monitoring. User context can change at any time and it is crucial for the application to detect those changes promptly. This requirement is difficult to accommodate due to limited smartphone resources, particularly the battery resource. Moreover, despite significant advances in smartphone processing power, context computation latencies remain prohibitively high for several interesting applications such as cognitive assistance [17]. For these reasons, users tend to avoid using context-aware applications.

S. Sigg et al. (Eds.): MobiCASE 2015, LNICST 162, pp. 143–160, 2015.
DOI: 10.1007/978-3-319-29003-4_9

Different offloading techniques have been proposed recently to address these issues of limited battery life and long computation latency. These techniques fall into two broad categories. In the first category, resource-hungry tasks are off-loaded to powerful servers residing in the cloud, leading to both computation speedup and energy efficiency. However, accessing the cloud incurs additional cost for the user in terms of cloud access fee and cellular data plan. In addition, access latency for cloud can be quite high. To address this, researchers are introducing cloudlets, acting as a middle-tier to bridge the gap between the mobile devices and the cloud [17].

In the second category, mobile applications use nearby mobile devices to share tasks, thereby minimizing the need for accessing cloud resources [6,10]. This helps with avoiding cloud and ISP charges, as nearby resources can be personal devices, or mobile devices of family members and coworkers. This also eliminates redundant sensing and computation, if several nearby mobile devices are interested in the same (shareable) context [10]. In addition, collaborative context monitoring extends sensor modalities and tackles the smartphone position problem [1]. However, this technique suffers from uncertainties due to the ad hoc nature of the network, lack of any apparent incentives for participation, security and privacy, varying device capabilities and device mobility.

It is clear that both of these offloading techniques have their pros and cons with one of them suitable for one scenario and the other one for a different scenario. At present, offloading solutions to cloud, cloudlet or nearby mobile devices exist in isolation. With proliferation of mobile devices, increased availability of (nearby) computing servers that can operate as cloudlets, and improved connectivity to the cloud, a highly likely scenario is one where a user has access to multiple computing resources whenever she/he needs to perform a context computation. Figure 1 illustrates four common scenarios in a typical user's (Alice) life. In the morning, Alice takes a bus to go to her work (*Bus scenario*). During her bus ride, she can perform collaborative computing with mobile devices of other bus riders. Later, in her work place (*Work Place scenario*), she can perform collaborative computing with mobile devices of co-workers as well as an office server (cloudlet) accessible within one network hop. During lunch time (*Lunch Break scenario*), Alice goes to a restaurant, where she can perform collaborative computing with mobile devices of other restaurant customers as well as a cloudlet provided by the restaurant. Finally, in the evening or on weekends, Alice goes for shopping in a mall with her family members and friends (*Shopping Mall scenario*), where she can use their mobile device for collaborative computing.

In this paper, we present a middleware framework called *Panorama* that enables mobile applications to reap the benefits of every computing opportunity (cloud, cloudlets, and other mobile devices) available at runtime. Panorama runs on multiple mobile nodes, and builds an optimized collaboration plan taking into consideration the users' performance objectives and the participants' preferences and constraints. A key challenge addressed in Panorama is to ensure an optimal partitioning of the computation task among available mobile devices, cloudlet, and cloud. Panorama considers important and practical constraints in

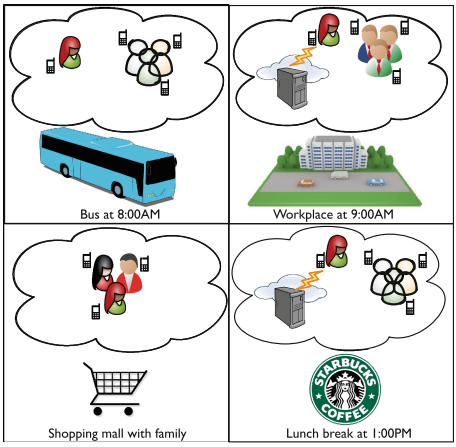

Fig. 1. Alice's typical day

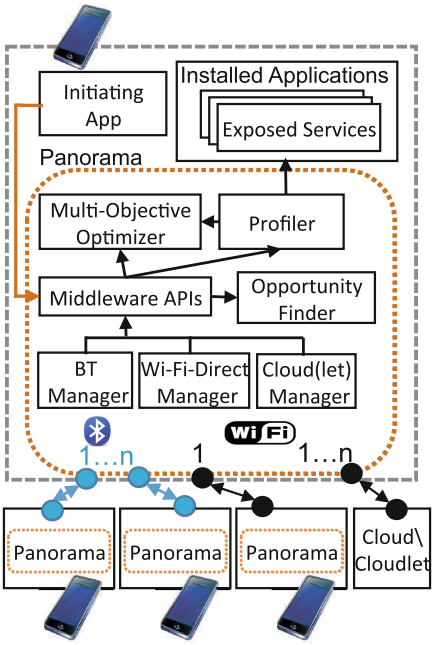

Fig. 2. Panorama's architecture

collaboration planning, such as the energy constraints of mobile devices, and their computation and communication capabilities. It also considers the costs involved in accessing nearby mobile devices vs. cloudlets/cloud. It also takes into consideration security, privacy, and trust relationship of the participating devices. At the heart of collaboration planning in Panorama is a versatile multi-objective optimization framework that takes into account various constraints of available computing opportunities and efficiently computes a collaboration plan that optimally trades off different performance objectives such as minimizing the overall cost, minimizing the energy consumption, and minimizing the execution time. Panorama provides support for both parallel and sequential (pipeline) task structures, two most common structures in context monitoring applications.

We have prototyped and extensively evaluated Panorama under a variety of scenarios in the presence of several different network configurations of mobile devices, cloudlet and cloud and under different device constraints. We have experimented with two representative context-aware applications: speech recognition (a parallel task) and ambiance sound monitoring (a sequential task). Experimental results show that Panorama can achieve both reduced computation time and decreased energy consumption while working within the constraints set by the collaborators, such as limits on the contributed energy, cost budget, and privacy requirement. Experimental results also show that Panorama is expressive and flexible in realizing different tradeoffs between completion time, energy consumption, and/or cost. Panorama is completely automated with no

user intervention needed after installation. A device with Panorama can automatically join a collaboration network when needed, and run tasks that are suitable for it.

2 Design

2.1 Overall Architecture

Panorama is designed according to the current mobile application development standards, and can be easily adopted without incurring much change to the current mobile software stack. It provides APIs to allow applications to discover nearby devices, cloud and cloudlets, build a network, and delegate tasks to them. It also provides APIs to allow other mobile devices to discover local resources and to accept task delegation. The overall architecture of Panorama is shown in Fig. 2. A device acts as an initiator and triggers the network creation phase when it needs to compute a costly context and is looking for collaborators. Panorama's design supports diverse network interfaces like Bluetooth, Wi-Fi Direct, and connections with IPs where cloudlets/cloud reside. Bluetooth standard allows creation of Piconets where the initiating device connects to multiple devices in a star topology. The initiating device can connect to another device using WiFi-Direct and to previously defined IPs for cloudlets/cloud.

Panorama is implemented as a background service that exposes the middleware APIs to applications looking for collaboration opportunities. The main component is the *middleware APIs component*. This component contains the APIs that the applications can call to use the framework's services. The *Bluetooth Manager* and *Wi-Fi Direct Manager* components implement technical details of short-range communication channels. Panorama defines the behavior that every communication channel must provide to support task collaborations like searching for other devices, connecting to other devices, accepting connections from other devices, accepting and responding to resource inquiry messages, and finally accepting and processing task delegation. Panorama currently supports two communication interfaces with other mobile nodes: Bluetooth and Wi-Fi Direct. With this design, it would be easy to plug in new communication interfaces (e.g., NFC or ZigBee) without any significant change in Panorama.

The *Cloud(let) Manager* component implements technical details of connecting with cloudlets/cloud. Currently, we pre-configure the IP addresses where the implementation for specific context monitoring task exists. However, we expect that network resource discovery techniques can be utilized to discover cloudlets/cloud efficiently.

The *Profiler* component gathers collaboration-relevant information about the mobile node Panorama is running on and provides it to other nearby mobile devices through Panorama's APIs. The initiating node delegates different context sub-tasks to different devices based on the profiler information. Currently, this component provides two types of information: available set of services on the device along with their performance metrics and constraints of the mobile device. Available services here refer to context derivation code that applications

expose so that other mobile devices can execute their context-aware task on the device. In the current design, the required code should be available on the device before collaboration can take place. Device constraints include energy quota, time quota, and incentives, etc.

The *Multi-Objective Optimizer* component employs multi-objective optimization to find the best collaboration plan that conforms to the device constraints and achieves the initiator's objectives. Currently, we optimize for energy consumption, execution time, and cost. However, due to its flexibility, the optimization model can be easily extended to accommodate for other parameters when required. Details of this component are discussed in Sect. 2.3. Finally, the *Opportunity Finder* component performs regular scanning for nearby devices using Bluetooth and maintains a list of recently discovered devices to be utilized whenever collaboration is required.

2.2 Application Partitioning and Profiling

Collaborative context monitoring involves changing the application execution model from standalone execution on a single mobile device to distributed execution on multiple mobile devices, clouds and cloudlets. This requires partitioning the application and making the code for calculating context available on collaborating nodes before the collaboration. Panorama's design requires the context code to be available on other mobile devices before the collaboration. This design choice is reasonable since Panorama targets shareable contexts that will be of common interest. For example, a programmer will write a speech recognition component that takes an audio as input and returns text as a result. This component can then be made available for other applications running on the device as well as for nearby collaborators by being exposed as a service through the operating system. Note that this design choice is consistent with research in the field that proposes contextual data units [4] and envisions sharing them among collaborators [8]. Technically, Panorama utilizes the Android service component to support this design (see Sect. 3). For servers, we envision a future where popular shareable context is exposed by server APIs analogous to web APIs.

For application profiling, Panorama tracks the required execution time and energy consumption for exposed services and provides this information through the API to other mobile devices. Panorama uses information gathered from previous invocations to build a linear regression model similar to the work in [7] that predicts the execution time and energy consumption for future tasks delegated to the node. We choose to use file size as the input to this regression model. This choice has proven accurate for speech recognition tasks in our current implementation. The energy profiling is done manually by taking measurements from an external power source. This workaround solution is required due to the lack of an accurate API that exposes energy consumption of the device to solutions like Panorama.

2.3 Optimization Models

Consider a system where n devices, indexed by $i = 1, \cdots, n$, collaborate on a certain task with a total workload of w. Let $x = (x_1, \cdots, x_n)$ denote the

allocation of the task, with device i being allocated an amount $x_i \geq 0$ of workload. Obviously, $\sum_{i=1}^{n} x_i = w$. Denote by e_i the energy consumption for processing one unit of workload by device i. We assume that each device i has an energy budget b_i that it is willing to expend for collaboration, i.e., $e_i x_i \leq b_i$. Denote by c_i the payment received by device i for processing a unit of workload, and B the initiator's total budget on payment. So, $\sum_{i=1}^{n} c_i x_i \leq B$. We further assume that each device i takes an amount f_i of time to process one unit of workload. We aim to minimize both the energy consumption and the completion time, which is formulated as the following multi-objective optimization problem:

$$\min_{x \succeq 0} \ (\text{w.r.t } R_+^n) \ (\sum_{i=1}^{n} e_i x_i, \quad \max_i \{f_i x_i\}) \tag{1}$$

$$\text{s.t.} \ \ e_i x_i \leq b_i, i = 1, \cdots, n \tag{2}$$

$$\sum_{i=1}^{n} x_i = w \tag{3}$$

$$\sum_{i=1}^{n} c_i x_i \leq B. \tag{4}$$

Introducing a weight $\gamma \geq 0$ to specify the tradeoff between energy consumption and execution time, we can solve the above problem by scalarization, which can be reformulated as a linear program (LP):

$$\min_{x \succeq 0} \ \sum_{i=1}^{n} e_i x_i + \gamma t \tag{5}$$

$$\text{s.t.} \ \ e_i x_i \leq b_i, i = 1, \cdots, n \tag{6}$$

$$f_i x_i \leq t, i = 1, \cdots, n \tag{7}$$

$$\sum_{i=1}^{n} x_i = w, \quad \sum_{i=1}^{n} c_i x_i \leq B. \tag{8}$$

A larger (smaller) γ means a higher preference/priority on short completion time (low energy consumption). In practice, the value of γ is set based on the initiator's preference.

Notice that in the above optimization problems, we impose a hard constraint on the initiator's budget; see Eq. (4). But we can also make the payment an objective to optimize. For example, we can optimize both the initiator payment and the completion time under the energy budget constraint, i.e.,

$$\min_{x \succeq 0} \ \sum_{i=1}^{n} c_i x_i + \gamma t \tag{9}$$

$$\text{s.t.} \ \ e_i x_i \leq b_i, i = 1, \cdots, n \tag{10}$$

$$f_i x_i \leq t, i = 1, \cdots, n \tag{11}$$

$$\sum_{i=1}^{n} x_i = w. \tag{12}$$

The above modeling framework can be easily extended to incorporate different performance objectives or concerns. For example, for certain reason such as privacy concern, we may require certain portion of workload u to be processed at a subset $i = 1, \cdots, m$ of devices such as those that can be trusted. This can be ensured by imposing an additional constraint $\sum_{i=1}^{m} x_i = u$ to the above optimization problems.

In practice, the values of e_i and f_i can be measured/estimated as described in Sect. 2.2. The value of c_i will be determined by each device/collaborator based on its resource scarcity or abundance as well as incentive. The total number n of collaborating devices is usually a small number less than 10, resulting in a small LP problem. The LPs (5), (6), (7), (8), (9), (10), (11) and (12) can be solved on smartphone using existing LP solvers, e.g., Apache for Java, in tens of millisecond. We have implemented a customized solver especially for Panorama.

2.4 Discovery Protocol

For collaboration, Panorama needs to know what devices are available nearby and how long those devices are expected to stay within the collaboration range. Under Bluetooth v3.0, Panorama needs to scan its surroundings regularly, which results in significant energy overhead. To minimize this overhead, Panorama utilizes adaptive scanning based on discovered number of peers as described in [9]. The new Bluetooth Low Energy (BLE) protocol provides lightweight mechanism for broadcasting device capability beacons in a connectionless mode called advertising. Panorama can leverage this feature for efficient service discovery and switch to classic Bluetooth for sending files at higher rates. Unfortunately, none of our Android devices support BLE peripheral mode required for advertising. We plan to incorporate BLE in Panorama as part of future work.

3 Implementation

We have implemented Panorama as a background service on the Android platform, which can be installed as a user-space application. After installing Panorama, context-aware applications running on the same mobile device can use Android IPC to call its APIs. Panorama's simple interface has a start/stop button that can be used by users to indicate their willingness to engage in collaboration. Once Panorama is started, it can automatically accept Bluetooth connections from co-located devices that also run Panorama. Bluetooth connection between Panorama copies running on different mobile devices can take place automatically without user intervention. In order to support this requirement, Panorama uses a specific Bluetooth UUID as an identifier and connects using Bluetooth insecure channel. This design allows for automatic creation of connections with co-located devices which is a mandatory requirement for systems such as Panorama to work. However, it introduces security risk from an adversary with access to the UUID. Techniques to secure mobile ad hoc networks such as reputation systems [2] and secure key management [3] can be

employed to secure Panorama. We are considering implementing these techniques in Panorama as part of future work. Panorama also utilizes Wifi-Direct as an additional communication channel. However, the connection has to be accepted by the receiving party since this is the only supported scheme on Android implementation of Wifi-Direct. When the communication network is established, devices can discover resources and delegate tasks to each other.

3.1 Panorama's Programming Interface

Method signatures of Panorama's APIs are defined using the Android Interface Definition Language. In order for third-party applications developers to call Panorama's APIs, they will need to include a copy from the .aidl file in their application package and bind to Panorama's middleware service. Table 1 lists method signatures from this file. The group of APIs handling Bluetooth, Wifi-Direct and Cloudlet perform the required functions for creating the network using the communication channels. To create the Bluetooth network, a device checks the freshness of recently discovered devices list using *get_bt_devices*, then invokes the *create_piconet* API. The latter API automatically connects to nearby devices running Panorama. An application can also connect to a device through WiFi-Direct and a cloudlet/cloud server through Wi-Fi using the Wifi-Direct APIs and cloudlet APIs respectively. For generic APIs, the *discover_nw* API sends a discovery message for all connected devices with the required service name. Panorama's design utilizes the Android service component for code discovery. That is, application developers will write context derivation code in an Android service and expose it under a unique identifier through the Android OS. Accordingly, whenever a device receives a discovery message, it triggers the *get_local* API, which checks whether the required service is installed on the device by checking exposed services against the provided *service_name* string. After gathering information about connected devices, the application can trigger the optimization process and perform the collaboration using the *perform_optimization* and the *execute_optimization* APIs respectively. Upon receiving task delegations, collaborating devices can process their portions of the task using the *process_local* API.

Table 1. Method signatures from Panorama's aidl file

Generic APIs	Bluetooth, Wifi-Direct & Cloudlet APIs
`List<Device> discover_nw(service_name)`	`List<Device> get_bt_devices`
`Device get_local(service_name)`	`create_piconet`
`Plan perform_optimization(task)`	`start_wd_discovery`
`Result execute_optimization`	`List<Device> get_wd_devices`
`Result process_local(task)`	`connect_wd(device_name)`
	`connect_to_cloudlet`

In the description above, we have provided detailed APIs from Panorama for clarity. However, we note that these APIs can be combined to shield collaboration logic from the application logic and make task delegations happen automatically. For example, we have implemented an API that both connects to nearby devices and discovers them for a specific service in one step.

3.2 Experiment Testbed

To evaluate the utility of Panorama, we have implemented two context-aware applications representing two different application structures: parallel structure and pipeline structure. A speech recognition application that is based on PocketSphinx [15] to perform speech recognition from a dictionary represents a parallel task. This task is computation-intensive, making it a good candidate for collaboration. For the pipeline structure, we implement the sound ambiance monitoring task from [12]. We define three stages and run them in three different Android services components to distribute the application. First, an audio recording stage, which represents the sensing stage. Second, a stage that calculates FFT for the audio window and generates features to classify sound as either music or speech. Finally, a third stage that takes the FFT as input and generates MFCC vector, which is then used to identify the gender of the speaker. This is only used when the sound is detected as speech. We also implement both applications using Java to be able to run them on cloudlets and clouds. We integrate these applications with Panorama and run experiments on four Android mobile devices running different versions of the Android OS and a laptop to emulate a cloudlet compute box. We also rent an Amazon EC2 server to use for experiments involving the cloud. The Android devices used are Galaxy S4, Galaxy Note, Galaxy Tab 3, and Galaxy Nexus. Galaxy Note, Galaxy Tab 3 and Galaxy Nexus have dual-core processor while Galaxy S4 has a quad-core processor.

4 Evaluation

4.1 Methodology

We have evaluated Panorama for a variety of scenarios under different collaboration opportunities, resource restrictions, and incentives. The experimental settings are chosen to reflect real-life scenarios that a system like Panorama may face. Collaboration opportunities include cloudlets/cloud as well as multiple mobile devices belonging to the user, his/her friends and family members, and/or strangers. The initiator may have different objectives, and the collaborators may have different constraints in terms of energy, time, cost, and privacy.

The execution time reported in the experiments is the total time to execute the required task using a collaboration, including the time for connecting with other devices or cloudlet, devising an optimal task partitioning plan, shipping subtasks, and gathering the results back. The energy consumption reported is

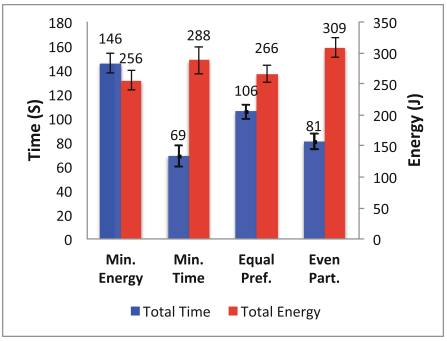

Fig. 3. Tradeoff between energy consumption and execution time by Multi-objective optimizer (3 collaborators).

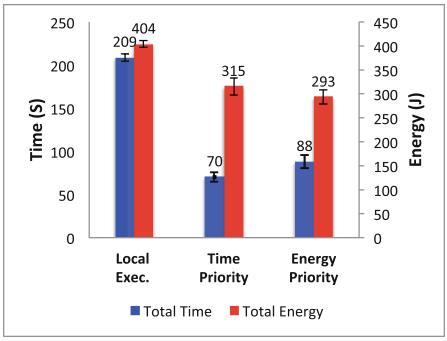

Fig. 4. Impact of privacy on energy and time optimization (3 collaborators).

the sum of energy consumed in all mobile devices (not cloudlet) that participate in the collaboration. It takes into consideration the energy consumed in all stages from the creation of network for collaboration to the gathering of the results back to the initiating device. To measure energy consumption of different activities, we log the device electric current drain indicated by the power supply unit, and then subtract the average current drain observed before the measured activity starts in order to obtain the additional current drain caused by the collaboration activity. We multiply the additional current with the voltage applied at the battery terminals of the mobile device to get the instantaneous power consumption of the activity in Watt and then integrate it over time to obtain the energy consumption of the activity in Joule. We ensure that there are no other applications running in the background. We repeated each experiment five times and report the average of the measurements from these five trials. We also report standard deviation, which is rather low in all experiments.

4.2 The Utility of Multi-objective Optimizer

We use speech recognition to evaluate the adaptability of Panorama to different participant preferences and resource restrictions. Speech recognition is a good candidate for task collaboration because of its compute-intensive nature, and is used as an example context-aware task that can be distributed in parallel. We envision Panorama integrating with context-aware applications that require general speech recognition. In the experiments, we consider a scenario where an audio file of 4 MB is recorded and requires performing speech recognition.

Tradeoff Between Energy Consumption and Execution Time. We first demonstrate that Panorama provides support for appropriate partitioning to achieve the desired tradeoff between energy consumption and execution time. Such a tradeoff is needed in a Bus or a Shopping Mall scenario described in

Fig. 1, where only mobile devices may be available for collaboration and the user does not have access to a power source. Due to the lack of access to power source, the remaining battery level dictates how important it is to minimize energy consumption during computation.

In this experiment, we assume that there are two mobile devices available in the user's vicinity in addition to the user's own mobile device. We also assume that the speech data does not contain any sensitive information and so privacy is not a factor in optimization. In the next experiment, we will take into consideration the privacy concern when certain parts of the speech data are sensitive in nature. Recall from Subsect. 2.3 that different choices of weight γ correspond to different tradeoffs between energy consumption and execution time. Figure 3 shows the comparison between $\gamma = 0$ (minimize energy consumption, corresponding to a situation with relatively low remaining battery level), $\gamma = \infty$ (minimize execution time, corresponding to a situation with relatively high remaining battery level), and $\gamma = 1$ (equal preference over energy and time, corresponding to a situation with moderate remaining battery level). We also contrast this with a naive partitioning strategy that divides the task evenly over all the participants. Figure 7 shows the corresponding partitioning of speech data for each of the collaborators for these different cases. For brevity, we report in the same figure the file partitioning of speech recognition tasks for other experiments as well that are described later.

We see that in the case of minimum energy (low remaining battery level), bigger chunks of file are sent to the participating device with low energy consumption without any emphasis on exploiting parallelism to achieve computation speedup, leading to slow execution. The opposite trend is observed for the case of minimum time (high remaining battery level). Here, the total execution time is minimized at the expense of increased energy consumption. The equal preference case represents a compromise between the previous two cases, with execution time and energy consumption in between those of these two cases. Finally, the even partitioning scenario is able to exploit parallelism to achieve a good performance in time. However, it consumes the most energy because of the lack of any planning in this aspect.

Notice that in the previous experiment, the task distribution is same whether the user is in a Bus scenario or a Shopping Mall scenario, because the speech data does not contain any sensitive information. We now consider the case where some parts of the speech data contain sensitive information (1.5 MB out of the total 4 MB is sensitive). In the Bus scenario, since the mobile devices other than the initiator's are untrusted, the sensitive parts of the data cannot be shipped to them. As a result, only 2.5 MB of (non-sensitive) speech data is available for collaboration in this case and the remaining 1.5 MB of (sensitive) data must be processed at the initiator's device. Figure 4 shows the results of this case. We see that in both time and energy priority cases, the achieved time gain from exploiting other mobile nodes is more than 50 %. In the case of energy priority, 27 % energy is saved compared to local execution due to shifting of the (non-sensitive) portion of the task to a more efficient mobile node. Also, when we

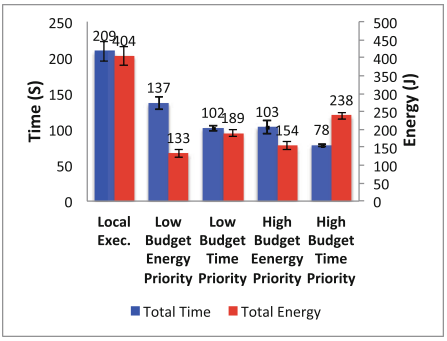

Fig. 5. Impact of the initiator's willingness to pay (3 collaborators & 1 personal low end device).

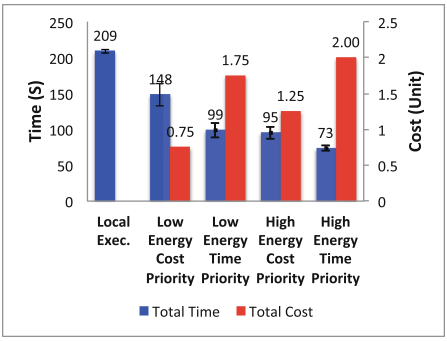

Fig. 6. Impact of the collaborators' energy budget (3 collaborators & 1 personal low end device).

compare the energy priority and time priority cases, we see that Panorama is able to devise the best plan for executing the non-sensitive part of the file. For the time priority, Panorama divides the file efficiently (see Fig. 7) to save 20 % of time when comparing to the energy priority case. As for the energy priority, Panorama sends the file chunks to the more energy efficient device thereby saving 7 % more energy. For the Shopping Mall scenario, since all devices are trusted, the presence of any sensitive data does not make any difference in task distribution. The results are same as those reported in Fig. 3.

Impact of Collaborator Constraints. We now evaluate the capability of Panorama to optimize for different objectives under different constraints specified by the participants. Consider the Bus or the Shopping Mall scenario with three mobile device collaborators and an additional mobile device (which is a tablet in the experiment) that belongs to the initiator. This corresponds to a situation where the initiator has two mobile devices, one of which is a low end device that has poor performance but is "free" in terms of cost and energy. The user may want to use the low end device to save time and energy or meet a cap on cost. We consider a situation where the initiator pays the collaborators, and the payment is proportional to the amount of work done.

We first consider a scenario in which the initiator has a budget on the total amount she is willing to pay. We experiment with two budget levels: a low budget of 2 units of payment and a high budget of 4 units of payment.[1] We consider two situations, one that aims to minimize energy consumption, and the other that aims to minimize execution time. The preference/priority on energy or time is represented by choosing a small or large weight γ in problem (5), (6), (7) and (8). Figure 5 shows the results of this experiment, and the corresponding partitioning can be found in Fig. 7.

[1] Notice that here the word "payment" is used in a general sense. It can be a monetary payment, or virtual payment such as credit for reputation.

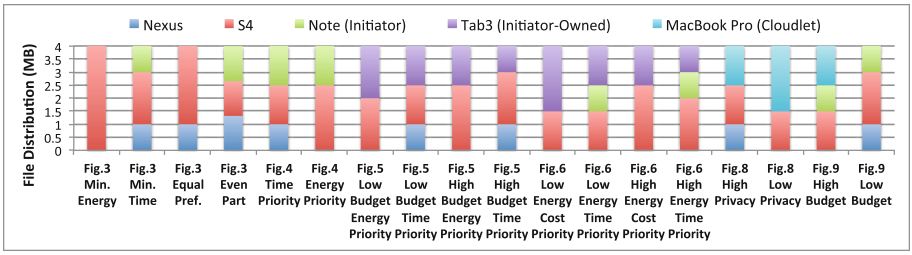

Fig. 7. File distribution for the 4 MB task in experiments of Figs. 3, 4, 5, 6, 8 and 9.

We see that, with low budget and if energy is of high priority, Panorama sends most of the task to the free initiator-owned low end device, in order to save energy in other devices and meet the payment cap while incurring a long execution time. When execution time is of high priority, Panorama sends the task more to the devices that are fast, which leads to 25 % reduction in time while costing much more energy. On the other hand, with high budget, if time is of high priority, Panorama achieves a large reduction in time by shifting larger portion of job to fast but costly devices. When energy is of high priority, the energy consumption goes up compared to the low budget case. This is because the large reduction in time from faster computing allowed by higher budget compensates the increase in energy consumption. As expected, compared to local execution, collaboration reduces execution time and saves energy.

We now consider a scenario where the collaborators have a restriction on the amount of energy they are willing to expend for collaboration, and investigate the tradeoff between execution time and initiator's cost/payment under different energy budgets; see problem (9), (10), (11) and (12). Figure 6 shows the results of an experiment with a low, 100 Joules energy budget for each mobile device, and a high, 200 Joules energy budget for each device; and the corresponding partitioning of speech data can be found in Fig. 7. We see that, compared with low energy budget case, high energy budget leads to shorter execution time when comparing both the cost and time priority cases to their corresponding low energy cases. This is because higher energy budget allows for longer use of faster devices. Also, notice that, with low energy budget and if the cost is of high priority, Panorama sends larger portion of the task to the free initiator-owned low end device, resulting in large execution time and the lowest cost. We also compare with the case of local execution. As expected, collaboration leads to shorter execution time while incurring cost as a result of utilizing other nodes.

Presence of Cloudlets. We now consider scenarios where a cloudlet is available in addition to some mobile devices for collaboration, as in the Work Place and Lunch Break scenarios shown in Fig. 1. In a Work Place scenario, a user has high trust in the available mobile devices as they belong to her/his co-workers. In addition, in some Work Place scenarios, the user may also trust the cloudlet,

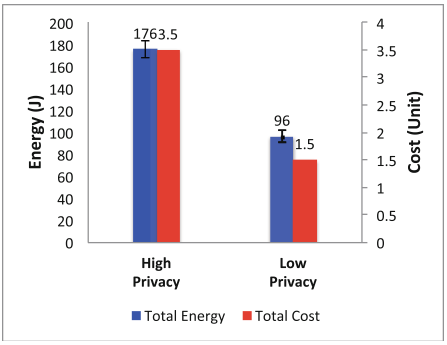

Fig. 8. Impact of privacy requirement of the task (3 collaborators & cloudlet).

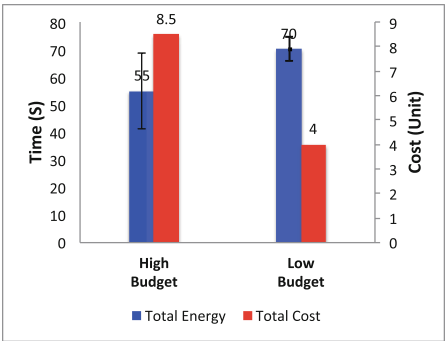

Fig. 9. Impact of collaborators cost budget requirement of the task (3 collaborators & cloudlet).

while in other cases, she/he may not trust it. On the other hand, in the lunch break scenario, neither the cloudlet nor the other mobile devices may be trusted.

In the first experiment reported here, we consider a Work Place scenario that involves three mobile devices and an untrusted cloudlet. An audio file is divided into sensitive and insensitive parts. We consider two cases here: a high privacy case with 2.5 MB out of the total 4 MB file marked as sensitive, and a low privacy case with only 1.5 MB marked as sensitive. As shown in Fig. 8 (and Fig. 7), imposing higher privacy leads to higher energy consumption and higher cost. This is because only a small portion of the speech data is sent to the faster and energy-cost-free cloudlet. For lower privacy case, a much larger portion of the speech data is sent to the cloudlet, thus reducing energy consumption and execution time. We conduct two additional experiments for the cases when the cloudlet is trusted and when there is no sensitive data in the audio file. In both cases, Panorama offloaded almost the entire file to the cloudlet. This is because the cloudlet is significantly faster than the mobile devices and does not contribute to energy overhead. We do not report the results of these experiments here due to space limitation.

Next, we consider the Lunch Break scenario where both the cloudlet and the collaborating mobile devices are untrusted and there may be a cost associated with using them. In such a situation, the user has no choice other than executing sensitive parts of the task locally. Yet, Panorama can still optimize for the remaining non-sensitive portion of the task to devise an efficient plan, thereby, minimizing the burden on the initiator as much as possible. Figure 9 reports the results of an experiment where the user would like to process 4 MB of insensitive speech while minimizing the execution time (i.e., $\gamma = \infty$); and the corresponding partitioning of data can be found in Fig. 7. In contrast to the previous experiment, we gave the cloudlet here a higher cost of 4x compared to 1x for other nodes. In the case "high-budget," the user allocates a budget of 10 units to the task, whereas, in the "low-budget" case only 5 units are allocated. We see from the figures that when the budget is high, Panorama was able to shift a big portion of the task to the cloudlet achieving better computation speedup

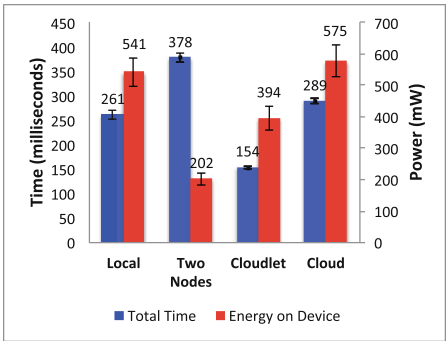

Fig. 10. Benefits of collaboration for sound ambiance monitoring.

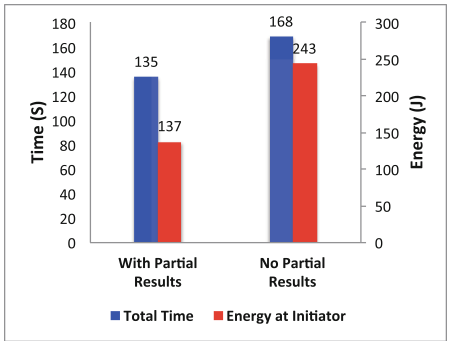

Fig. 11. Impact of leaving node on Panorama's performance.

compared to low budget scenario. However, the low budget scenario was not as slow as we expected when compared to the high budget scenario. The reason is that Panorama was able to exploit parallel execution (see Fig. 7 for file distribution) with other collaborators within the allocated low budget without worrying about energy consumption since it is not considered in this scenario.

4.3 Benefits of Collaboration for Sequential Tasks

To demonstrate the utility of Panorama in handling sequential task structures, we have implemented the sound ambiance monitoring application proposed in [12], and employed Panorama to enable collaboration. The results of the collaboration experiments are then compared to the local execution on a single device with Panorama turned off. For collaboration, we conducted three experiments. In the first experiment, the initiator collaborates with two other mobile devices, and in the second and third experiments, the initiator collaborates with a cloudlet sitting on the same network and a cloud server accessed through a Wi-Fi Internet connection. Recall from Sect. 3.2 that the sound ambiance monitoring task can be viewed as a pipeline consisting of three subtasks. Those can be split among collaborators. From Fig. 10, we see that in case of collaboration with two other mobile devices, there is a reduction in power consumption at the initiator's device from 541 mW to 202 mW, a more than 50 % energy reduction by delegating the calculation to the other collaborator. However, this comes at the cost of an increased completion time from 261 ms to 378 ms. Completion time here is the time between when the audio recording is completed and when the gender classification result has arrived at the initiator from the collaborator (or calculated locally in case of local execution). The increase in completion time is due to the time needed to set up the collaboration task and transfer the data and results between the collaborators.

Interestingly, the second experiment involving cloudlet does not show an increase in completion time. Instead, a time saving of 40 % is achieved in addition to the energy saving of 27 %. Here, the overhead introduced by Panorama is offset by the significant gain in execution time when delegating the compute-intensive

parts to the cloudlet. We also ran the same experiment to engage in a collaboration with an Amazon EC2 sever over a Wi-Fi Internet connection. The achieved result is worse both in terms of energy and time when compared to the cloudlet case. However, when we compare this result to a nearby node collaboration, we see that cloud can be a better alternative, depending on the intensity of the task, in terms of time while the opposite was true for energy.

4.4 Handling Mobility

There are two main challenges when it comes to handling node mobility: how to detect that a node is moving away, and how to ensure smooth migration of unfinished sub-task to other devices when a node leaves. For detection, we use a method proposed in [13] where we sense the accelerometer during collaboration to detect the starting of a physical activity as an indicator for a collaborator eventually leaving the scene. Once Panorama detects such activity, it sends a message to the initiator to handle mobility. We use accelerometer due to its relatively cheap energy cost and the fact that it can detect mobility promptly. Handling of interrupted collaborations depends heavily on the nature of the computation. In some situations a partial result can be migrated back to the initiator, whereas in others the whole computation need to be reprocessed. We performed two experiments with the aforementioned cases and report the results in Fig. 11 to reflect the impact of each on performance. The experiment considers a scenario where a node delegates a 4 MB task equally to two other nodes and one node moves away from the initiator. In the first case, we deliberately divided the received file and let the moving node finish the first 1 MB before moving away. Upon moving away, the node sends the partial result back to the initiator, which processes the remaining 1 MB locally. In the second case, we let the moving node report its movement without sending any partial results, so the initiator will process the whole 2 MB. Figure 11 reports the total time for completing the 4 MB task and the consumed energy at the initiator. As expected, the first scenario of partial result migration is better in terms of both energy and time when compared to the second scenario, since the initiator only needed to process half of the load assigned to the moving node. Notice that in our current implementation, when a collaborating node is leaving, the initiator picks up the unfinished work. We can also re-distribute the unfinished work among the remaining devices, which we plan to explore in future.

5 Related Work

Our work is closely related to [10,14]. However, [10] focuses on collaborative context monitoring between co-located mobile devices only, while Panorama leverages more opportunities by involving not only co-located devices but also cloudlets/cloud and performing an optimization to devise an optimal collaboration plan for the task. The main goal of [14] is to enhance the reliability of the application, while the goal of Panorama is to automate and optimize collaboration for continuous context monitoring.

The work in [18] studies generic computation offloading between co-located mobile devices, and presents three algorithms to serve three different possible applications' structures while taking into consideration connectivity in distributing jobs. The implementation in [18] is limited to a prototype that performs offloading between two devices only. Panorama's design involves more opportunities by including cloudlets/cloud in addition to co-located mobile devices. We also provide an extensive Android implementation and evaluate it on multiple mobile nodes and a cloudlet/cloud. The recent work in [8] focuses on building a conceptual model to facilitate context sharing between groups of mobile devices. Such model can be leveraged by Panorama to increase the chances of meeting peers and building more beneficial collaborations.

There are several vision papers that advocate the concept of collaboration among co-located mobile devices [6,13,21] and we have used some of their ideas to motivate our work. Also, a rich body of literature exists for augmenting smartphones with resources from cloud and cloudlets; see, e.g., [5,7,16,17]. The ideas in these works have helped in guiding our design.

6 Conclusion

Panorama is a middleware framework that addresses a key question in offloadling computations to nearby mobile devices and cloudlets/cloud: when should a device offload its context computing task and how? Panorama utilizes all available collaboration opportunities from co-located mobile devices and cloudlets/cloud, and devises a collaboration plan to optimize for and trade off different objectives such as minimizing execution time or minimizing energy consumption. The optimization algorithm considers limits set by participants such as contributed energy, paid incentives, and privacy exposure. Evaluation results show that Panorama is rather practical, is able to cope up with varying device constraints, and devises collaboration plans within those constraints to optimally trade off multiple objectives.

There are a number of future directions we plan to pursue. First, we plan to incorporate Bluetooth Low Energy in our opportunity discovery protocol. While none of the Android devices we test run in peripheral mode at present, we expect that Bluetooth-enabled smartphones will increasingly support Bluetooth LE. Second, we plan to expand on handling node mobility. At present, Panorama provides basic support for ensuring that the context computation task is completed despite some of the devices moving away. We plan to explore smart ways to efficiently cope with various mobility patterns. Third, a limitation in Panorama is to rely on collaborators to come up with privacy and efficiency requirements. An interesting research direction we plan to pursue is to automate the process of generating these requirements to enhance the practicality of Panorama. Finally, we plan to conduct user studies to evaluate Panorama in the real-world setting.

References

1. Alanezi, K., Mishra, S.: Enhancing context-aware applications accuracy with position discovery. In: Stojmenovic, I., Cheng, Z., Guo, S. (eds.) MOBIQUITOUS 2013. LNICST, vol. 131, pp. 640–652. Springer, Heidelberg (2014)

2. Buchegger, S., Le Boudec, J.-Y.: A robust reputation system for mobile ad-hoc networks. Technical report (2003)
3. Buttyán, L., Capkun, S., Hubaux, J.-P.: Self-organized public-key management for mobile ad hoc networks. IEEE Trans. Mob. Comput. **2**(1), 52–64 (2003)
4. Kansal, A., Liu, J., Chu, D., Zhao, F.: Mobile apps: it's time to move up to condos. In: HotOS (2011)
5. Chun, B.-G., Maniatis, P.: Augmented smartphone applications through clone cloud execution. In: HotOS (2009)
6. Conti, M., Kumar, M.: Opportunities in opportunistic computing. Computer **43**(1), 42–50 (2010)
7. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: MobiSys (2010)
8. de Freitas, A.A., Dey, A.K.: The group context framework: an extensibletoolkit for opportunistic grouping and collaboration. In: CSCW (2015)
9. Han, B., Srinivasan, A.: eDiscovery: energy efficient device discovery for mobile opportunistic communications. In: ICNP (2012)
10. Lee, Y., Ju, Y., Min, C., Kang, S., Hwang, I., Song, J.: Comon: cooperativeambience monitoring platform with continuity and benefit awareness. In: MobiSys (2012)
11. Lu, H., Frauendorfer, D., Rabbi, M., Mast, M.S., Chittaranjan, G., Campbell, A.T., Gatica-Perez, D., Choudhury, T.: Stresssense: detecting stress in unconstrainedacoustic environments using smartphones. In: UbiComp (2012)
12. Lu, H., Pan, W., Lane, N.D., Choudhury, T., Campbell, A.T.: Soundsense: scalable sound sensing for people-centric applications onmobile phones. In: MobiSys (2009)
13. Miluzzo, E., Cáceres, R., Chen, Y.-F.: Vision: mClouds-computing on clouds of mobile devices. In: MCS (2012)
14. Miluzzo, E., Cornelius, C.T., Ramaswamy, A., Choudhury, T., Liu, Z., Campbell, A.T.: Darwin phones: the evolution of sensing and inference on mobilephones. In: MobiSys (2010)
15. CMU PocketSphinx
16. Ra, M.-R., Sheth, A., Mummert, L., Pillai, P., Wetherall, D., Govindan, R.: Odessa: enabling interactive perception applications on mobiledevices. In: MobiSys (2011)
17. Satyanarayanan, M., Chen, Z., Ha, K., Hu, W., Richter, W., Pillai, P.: Cloudlets: at the leading edge of mobile-cloud convergence. In: MobiCASE (2014)
18. Shi, C., Lakafosis, V., Ammar, M.H., Zegura, E.W.: Serendipity: enabling remote computing among intermittently connectedmobile devices. In: MobiHoc (2012)
19. Wang, R., Chen, F., Chen, Z., Li, T., Harari, G., Tignor, S., Zhou, X., Ben-Zeev, D., Campbell, A.T.: Studentlife: assessing mental health, academic performance andbehavioral trends of college students using smartphones. In: UbiComp (2014)
20. You, C.-W., Montes-de Oca, M., Bao, T.J., Lane, N.D., Lu, H., Cardone, G., Torresani, L., Campbell, A.T.: Carsafe: a driver safety app that detects dangerous driving behaviorusing dual-cameras on smartphones. In: UbiComp (2012)
21. Zhang, W., Wen, Y., Wu, J., Li, H.: Toward a unified elastic computing platform for smartphones with cloud support. IEEE Netw. **27**(5), 35 (2013)