UNCERTAINTY-AWARE AUTONOMIC FRAMEWORK FOR RESOURCE MANAGEMENT IN MOBILE COMPUTING GRIDS

by

HARIHARASUDHAN VISWANATHAN

A dissertation submitted to the

Graduate School–New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Dario Pompili

and approved by

New Brunswick, New Jersey

May, 2014

ABSTRACT OF THE DISSERTATION

Uncertainty-aware Autonomic Framework for Resource Management in Mobile Computing Grids

By HARIHARASUDHAN VISWANATHAN

Dissertation Director:

Prof. Dario Pompili

Mobile platforms are becoming the predominant medium of access to Internet services due to the tremendous increase in their computation and communication capabilities. Also, as more and more of these mobile devices are coupled with in-built as well as external sensors capable of monitoring ambient conditions, biomedical and kinematic information, and location, they can provide spatially distributed measurements regarding the environment in their proximity. Cumulus, a mobile computing grid, which harnesses the heterogeneous sensing and computing capabilities of mobile devices in the field as well as that of servers in remote datacenters is envisioned. Cumulus can be exploited to enable innovative mobile applications (defined by workflows) that rely on real-time in-situ processing of sensor data.

However, enabling applications that require real-time in-the-field data collection and processing using mobile platforms is still challenging due to the following concerns: the inherent uncertainty associated with the quality and quantity of data from mobile sensors as well as with the availability of mobile computing resources in the field, security, and privacy. The goal of this research is to design and develop a unified uncertainty-aware (robust), secure, and privacy-preserving framework for data and computing resource management in the Cumulus in order to enable execution of mobile application workflows in real time and in situ and, hence, to generate actionable knowledge from raw data within realistic time bounds.

In order to achieve the stated goal, an autonomic (self-organizing, self-optimizing, and self-healing) middleware that aids in the organization of the sensing, computing, and communication capabilities of static and mobile devices in order to form Cumuli is proposed. As the relevance of the output of workflows rely heavily on the quality and quantity of raw data coming from the underlying sensing infrastructure as well as on the computing resources available to execute them in real time, a unified data and computing resource management mechanism for data- as well as task-parallel applications is proposed. Finally, `Maestro`, a robust, secure, and privacy-preserving framework for concurrent mobile application management in Cumulus is proposed. The proposed framework is evaluated using experiments on a prototype testbed as well as through simulations on a purpose-built Java$^{\text{TM}}$-based simulator.

Acknowledgements

Firstly, I would like to thank my advisor, Dr. Dario Pompili, an extraordinary researcher as well as a wonderful mentor. I am immensely indebted to him for his faith in my abilities. His friendly demeanor, approachability, and an ever-eager readiness to tackle problems head on are qualities that I admire in him the most and I feel honored to have been associated with him.

I am greatly indebted to my friends and colleagues at the Cyber-Physical System (CPS) Lab and at the Center for Cloud and Autonomic Computing (CAC). Special thanks to Eun Kyung Lee, Parul Pandey, and Baozhi Chen with whom I share a special bond and have spent the better part of my PhD years at Rutgers. I would like to thank Drs. Manish Parashar and Ivan Rodero for their timely words of advise and encouragement throughout the course of my doctoral research work. My sincerest thanks to all the co-authors in my publications. It has been a great learning experience working with each and every one of them. I would also like to thank all CPS and CAC members for their help in the form of ideas, implementation, and constructive feedback.

I would like to extend my gratitude to Drs. Manish Parashar, Ivan Marsic, and Hang Liu for accepting to be the committee members in my dissertation defense. Special thanks to Ms. Noraida Martinez, Mr. John Scafidi, and Mr. Jim Housell of ECE department for their care throughout my stay at Rutgers. I thank all the faculty and staff of ECE department at Rutgers for their help and support.

I was lucky to have wonderful roommates in Rajesh, Dev, and Rajat and I cannot thank them enough for their love and encouragement for all my endeavors. Many thanks to Pravin and Mangesh, my seniors and friends at Rutgers for suggestions and feedback on my research. Last but not least, I thank all of my family for their undying love and support, which has allowed me to chase my dreams and aspirations.

Dedication

To Amma, Anna, Sundar, Sneha

&

my late grandparents

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

This dissertation describes an uncertainty-aware data and computing-resource management framework for real-time in-situ processing of ubiquitous mobile application workflows in mobile computing grids.

1.1 The Case for Mobile Grid Computing

The computation and communication capabilities of mobile hand-held devices such as smart phones, tablets, and laptops have improved tremendously due to the advances in microprocessor, storage, and wireless technologies. It has been projected that, by 2015, mobile devices will surpass wired desktop PCs as the most preferred medium of access to the Internet and, therefore, will significantly impact distributed computing paradigms that use Internet-connected devices (volunteered by their owners [1]) as a source of computing power and storage. Also, as more and more of these mobile devices are coupled with in-built sensors capable of monitoring temperature, humidity, atmospheric pressure, sound, luminescence, acceleration, orientation, and direction, multiple cameras, and Global Positioning System (GPS) receivers, they can provide spatially distributed observations regarding the environment in their proximity. In addition, advances in the field of wireless sensors has led to the development of compact sensor nodes (also called motes) capable of communicating with other mobile devices and of capturing a wide variety of sensor data from biomedical to kinematic.

The mobile computing revolution is characterized by two clearly visible trends. The first trend is the aforementioned increase in mobile devices' computational capabilities aimed at improving user experience. The second trend is the growing popularity of the cloud computing paradigm, which pushes processing and storage (required for running applications)

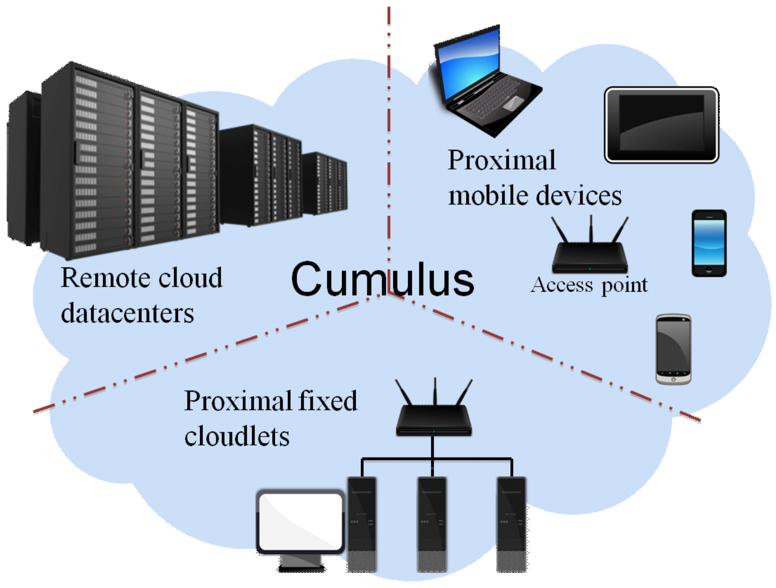


Figure 1.1: The spectrum of computing resources in a Cumulus – mobile resources in the proximity, fixed (cloudlets) computing resources in the proximity usually tethered to Wi-Fi access points, and cloud resources.

to remote servers on the Internet while retaining only a light front-end on the mobile device. The former is market-/demand-driven and is dictated primarily by users' purchasing behavior as well as by recent innovations in human-computer interaction; while the latter can be attributed to the gains that cloud computing provides in terms of lower infrastructure costs and reduced time-to-market for innovative applications (by offering infrastructure and platform as a service). These diverging trends have rendered the powerful mobile devices heavily under-utilized on average. We envision that the heterogeneous sensing, computing, communication, and storage capabilities of these under-utilized mobile devices in the field *as well as* that of computing and storage servers in remote datacenters can be *collectively* exploited to form a mobile computing grid (also referred to as a "loosely-coupled" mobile device cloud), which we call Cumulus (depicted in Fig. 1.1). The Cumulus can be harnessed to enable novel data- and compute-intensive mobile applications that rely on *real-time in-situ processing* of data generated in the field.

Mobile grid computing is a form of distributed computing where many networked mobile devices process massive amounts of locally generated sensor data in parallel. The response time, quality, and relevance of data- and compute-intensive mobile applications can be drastically improved through mobile grid computing. The emphasis on real-time in-situ processing arises out of the need to generate meaningful and actionable knowledge within realistic time bounds by processing raw data collected opportunistically from pervasive mobile sensing devices in the order specified by mobile-application workflows (the *raw data* $\rightarrow$ *information* $\rightarrow$ *knowledge* chain). The primary impediments to real-time in-the-field data processing are, 1) insufficient sensing and computing capabilities on individual mobile devices, which prevents them from producing meaningful results within realistic time bounds *in isolation,* and 2) the prohibitive communication cost and response time involved in enabling such data-intensive applications using the *wired-grid-computing* and/or *cloud-computing* approaches alone – in which computation and storage are offloaded to remote computing resources on the Internet.

Opportunistic discovery and exploitation of nearby compute and storage servers – referred to as *cyber foraging* [2] – was conceived to augment the computing capabilities of mobile hand-held devices in the field. Prior work in mobile cloud computing for cyber foraging has primarily focused on augmenting the computing capabilities of mobile devices in the field with *dedicated* and *trusted* computing resources, either situated remotely (in the *cloud* [3–5]) or proximally (in *cloudlets* [6, 7]). As cyber foraging was meant to convey a whole new pervasive computing paradigm based on the principle of "living off the land" [2], in this dissertation, we explore the feasibility of leveraging the computing and communication capabilities of other mobile devices in the field. In keeping with the broadest principles of cyber foraging, the Cumulus computing environment may be composed of (i) purely mobile resources in the proximity, (ii) a mix of mobile and fixed resources in the proximity, or (iii) a mix of mobile and fixed resources in the proximity as well as in remote datacenters as shown in Fig. 1.1. In this dissertation, we focus on the "extreme" scenario in which the Cumuli are composed *purely* of proximal mobile devices as it brings all the following concerns to the fore: robustness, security, and privacy. One or more of these concerns do not arise in the other scenarios and, hence, solutions developed for the extreme case will easily extend to the other cases.

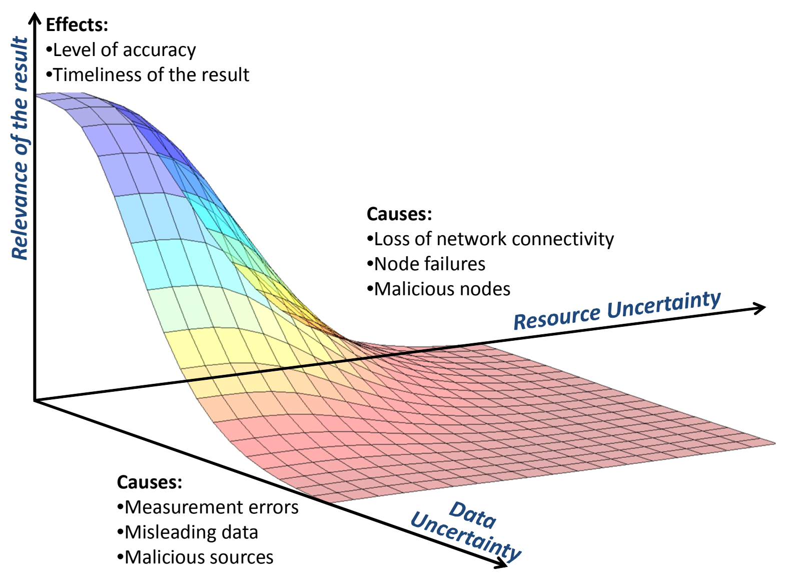


Figure 1.2: A sample path of uncertainty propagation in mobile application scenario. Data and computing-resource uncertainty, if unchecked, may propagate up the data-processing chain and have an adverse effect on the level of accuracy and timeliness of the final result.

1.2 Research Challenges

Mobile-application workflows are typically made up of several data collection, computation, and result generation tasks with pre-determined parallelism and order of execution. These workflows can be *data parallel* or *task parallel*. In mobile applications exhibiting *data parallelism*, the data is distributed across different parallel computing nodes that perform the same task while in applications exhibiting *task parallelism*, the parallel computing nodes may perform different tasks on the same or different data. The accuracy and timeliness of the results from mobile-application workflows rely heavily on the quality and quantity of raw data coming from the underlying multi-modal sensing infrastructure as well as on the computing resources available to execute them in real time. There are numerous research challenges associated with our envisioned approach – the execution of mobile application workflows in mobile computing grids – due to the *inherent uncertainty* associated with data quality and quantity as well as with the computing resource availability, *security*, and *privacy*.

Data and computing-resource uncertainty, if unchecked, may propagate up the data-processing chain and have an adverse effect on the level of accuracy and timeliness of the final result as shown in Fig. 1.2. Uncertainty in data quantity and quality can be attributed to intermittent network connectivity and availability of mobile sensors, measurement errors, and misleading as well as malicious data sources. Computing resource uncertainty can be attributed to intermittent device availability (due to unpredictable mobility), varying rate of battery drain, susceptibility to hardware failures, and lack of a priori knowledge about computational task (in the workflow) performance on different mobile hardware and software platforms. Note that a large amount of high-quality data does not guarantee good results as it increases the computational complexity and, hence, the computing-resource requirements. A small degree of uncertainty with computing resources (or data) can outweigh the benefits brought by reliable data (or computing resources). Hence, ensuring reliable mobile grid coordination and guaranteeing mobile application performance under uncertainty are significant research challenges.

Prior research efforts in mobile grid computing have aimed at integrating mobile devices into the wired-grid and cloud computing infrastructure mainly as service requesters. However, real-time in-situ processing of application workflows requires exploitation of mobile devices as service providers and uncertainty-aware resource management for reliable grid operation even in highly dynamic and unpredictable scenarios. Recently, a computing paradigm that has gained prominence is *approximate computing,* which identifies and captures the limitations arising out of either imperfect/partial data or insufficient computing resources to provide not-so-accurate yet meaningful results within specified deadlines. Prior work in the field of approximate computing has focused either on data uncertainty management or on computing resource uncertainty management in isolation to prevent the uncertainty from adversely affecting the result. However, approximate computing in the mobile grid computing domain requires a unified approach as it is affected by uncertainties arising out of both the sources. The goal of this dissertation is to design and develop a unified uncertainty-aware (robust), secure, and privacy-preserving management framework for data and computing resources so to enable approximate computing in mobile sensing and computing platforms and, hence, to generate actionable knowledge from raw data within realistic time bounds.

1.3 Contributions

In order to address the research challenges associated with ensuring reliable mobile grid coordination and guaranteeing mobile-application performance under uncertainty, in this dissertation, we make the following contributions.

1.3.1 Autonomic Middleware for Mobile Grid Computing

We proposed a robust autonomic middleware that organizes the sensing, computing, and communication capabilities of static and mobile devices in order to form an heterogeneous mobile computing grid [8, 9]. The role-based middleware is imparted with autonomic management capabilities, namely, self-organization, self-optimization, and self-healing. Uncertainty awareness is the defining characteristic of our proposed middleware. Specifically, our contributions include,

- A *role-based* service discovery and workflow management module, which imparts the self-organization capability for handling service request arrivals as well as for task distribution and management. This module also maintains long-term statistics regarding the dynamics of the underlying resource pool to minimize the effect of the uncertainties on application performance.

- A novel energy- and uncertainty-aware *resource allocation engine*, which imparts self-optimization capability for allocating the workload tasks optimally among the computing resources and to ensure application Quality of Service (QoS) even under uncertainties. This engine is designed for a generic class of data-parallel applications, which do not require any form of coordination among processing devices.

- The novel concept of *application waypoints* – which impart the self-healing capability – to monitor continuously the effect of uncertainties on application performance. Waypoints also eliminate unrealistic assumptions such as existence of accurate models for application performance (CPU cycles, memory, and storage used as well as the execution time) on different mobile hardware and software platforms.

1.3.2 Uncertainty-aware Mobile Application Workflow Management

We proposed a unified uncertainty-aware framework for management of data and computing resources in order to enable ubiquitous mobile-application workflows execution on mobile sensing and computing platforms [10]. The goal is to generate actionable knowledge from raw data while meeting end-user-specified requirements in terms of result accuracy and delay. Our proposed framework builds on the autonomic middleware and is capable of determining on the fly, 1) the quality and quantity of data to request from the data sources in the field along with 2) the most appropriate computational model (from a suite of models) to process the raw data, extract features, and generate actionable knowledge, and 3) the amount of computing resources to utilize from those available in the field. This capability enables us to control the propagation of uncertainty up the data-processing chain. While the resource allocation engine in the autonomic middleware is capable of handling only data parallel applications, this framework is capable of handling both data and task parallel applications. Specifically, our contributions include,

- A simple yet powerful *generalized workflow representation scheme* to construct structured multi-stage (or multi-level) data-processing chains (tasks and dependencies) for ubiquitous mobile applications, which are composed multiple parallelizable and/or sequential computational tasks.

- A procedure for *estimating the propagation of uncertainty* from raw data to the final result using *interval arithmetic* on confidence intervals in order to determine the appropriate quality and quantity of data to collect from the sensors in the field.

- Formulation of the problem of assigning parallel independent tasks to computing resources at every stage of the workflow as a *combinatorial optimization problem with multiple bottleneck objectives* (e.g., application execution time and fairness in battery drain at computing devices).

- A stage-wise threshold-based heuristic, `Fast M-CBP` *algorithm*, for solving the aforementioned combinatorial optimization problem with multiple bottleneck objectives in polynomial-time.

1.3.3 Robust, Secure, and Privacy-preserving Mobile Grid Computing

We proposed Maestro, a novel framework for robust, secure, and privacy-preserving mobile grid computing [11]. Often, multiple service requests are generated *simultaneously* in a Cumulus and, hence, tasks belonging to multiple workflows have to be allocated and executed on the Cumulus resources. The aforementioned complex task-allocation problem, however, also presents opportunities: there may be multiple duplicate service requests and multiple duplicate tasks that are common across different workflows. Maestro *deduplicates similar tasks across workflows* as it lends itself to minimization of duplication in services rendered, leads to efficient real-time in-situ processing of simplified workflows (with fewer tasks than before) as well as to better utilization of computing resources. After deduplication, the tasks of the simplified workflows have to be scheduled for execution on the Cumulus resources. Specifically, our contributions include,

- Dedup, a sub-graph matching technique for task deduplication among workflows to address the non-trivial research challenge of identification of task duplicates across workflows and the creation of simplified workflows.
- A robust replication-based task scheduling mechanism, which employs *controlled replication* of critical workflows tasks in order to bestow the self-protection capability in addition to the aforementioned self-healing using waypoints.
- A mechanism for *privacy-preserving computing* in the Cumulus through *controlled access to sensitive user data* via multiple levels of authorization.

We have performed thorough analysis and evaluation of our proposed solutions for mobile grid computing through experiments on a prototype testbed of Android- and Linux-based mobile devices as well as through simulations on a purpose-built Java^{TM}-based simulator.

1.4 Broader Impact

Augmentation of mobile devices' capabilities using mobile grid computing will enable novel *compute-* and *data-intensive* mobile applications spanning across multiple domains, from education to infotainment, from assisted living to ubiquitous healthcare. Such applications

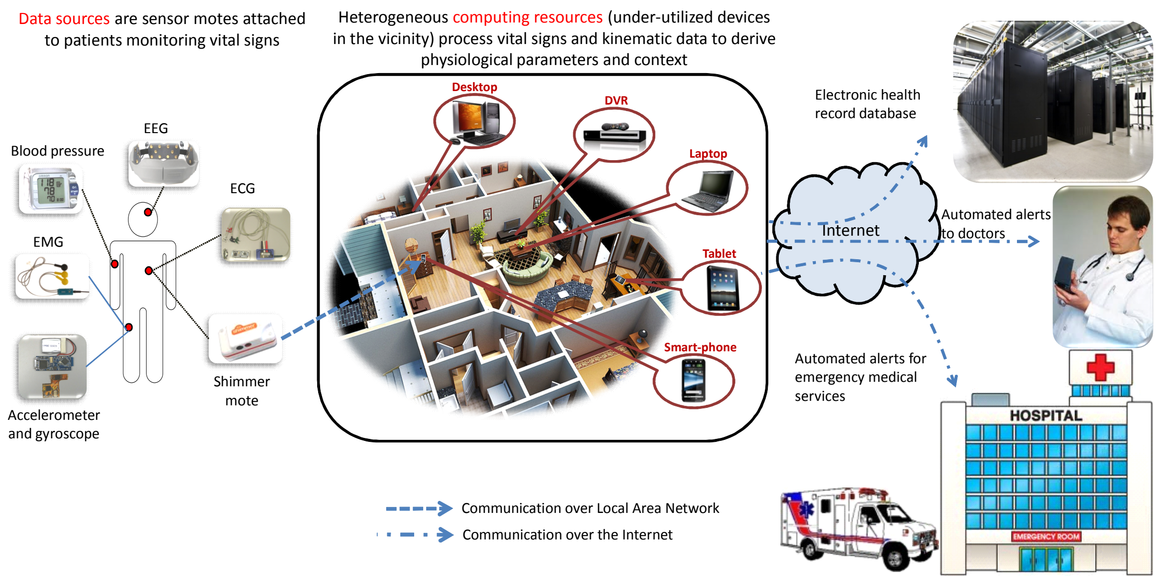

Data sources are sensor motes attached to patients monitoring vital signs
Heterogeneous computing resources (under-utilized devices in the vicinity) process vital signs and kinematic data to derive physiological parameters and context
Blood pressure
EEG
ECG
EMG
Accelerometer and gyroscope
Shimmer mote
Desktop
DVR
Laptop
Tablet
Smart-phone
Internet
Electronic health record database
Automated alerts to doctors
Automated alerts for emergency medical services
HOSPITAL
EMERGENCY ROOM
Communication over Local Area Network
Communication over the Internet

Figure 1.3: Mobile grid computing enabling ubiquitous health monitoring and care in the home or hospital setting.

include (but are not limited to) ubiquitous context-aware health and wellness monitoring of elderly (in home or hospital setting) and soldiers in the battlefield [9,12], distributed rainfall and urban flood-risk estimation, estimation of pollution level using distributed real-time air-quality measurements, distributed object recognition and tracking, and distributed user-generated multimedia search and sharing. Our ideas go beyond the application scenarios listed here and can be applied to any ubiquitous mobile application that relies on real-time in-the-field processing of locally generated sensor data. In the following, we elaborate on two of the aforementioned application scenarios.

Ubiquitous healthcare (home or hospital setting) Future ubiquitous healthcare systems will be characterized by i) pervasive vital sign monitoring using non-invasive sensors, ii) real-time processing of monitored data to derive meaningful physiological parameters, and iii) context-aware data- and patient-centric decision making. Our proposed autonomic resource management frameworks can harness under-utilized computing resources in the vicinity to support real-time processing of vital signs (using inherently complex physiological models) and to acquire context awareness (using machine-learning-based algorithms). However, there are multiple challenges associated with this vision of ubiquitous health care. Firstly, significant amount of preprocessing like noise rejection, data disambiguation, and

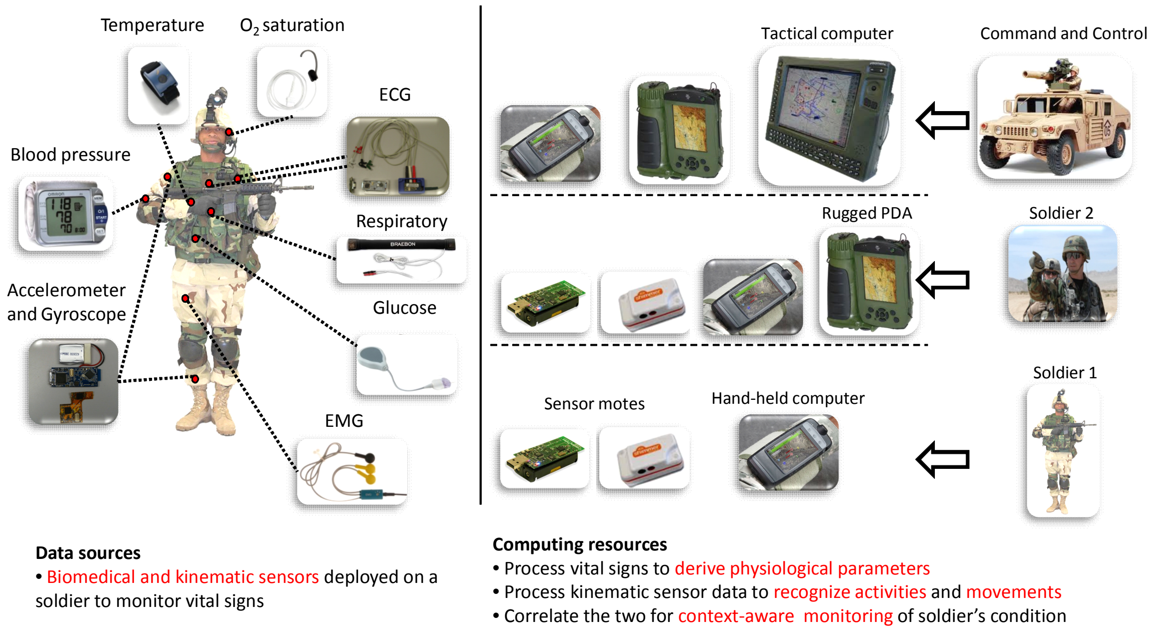

Temperature
O$_2$ saturation
ECG
Blood pressure
Respiratory
Glucose
Accelerometer and Gyroscope
EMG
Data sources
• Biomedical and kinematic sensors deployed on a soldier to monitor vital signs
Tactical computer
Command and Control
Rugged PDA
Soldier 2
Soldier 1
Sensor motes
Hand-held computer
Computing resources
• Process vital signs to derive physiological parameters
• Process kinematic sensor data to recognize activities and movements
• Correlate the two for context-aware monitoring of soldier's condition

Figure 1.4: Mobile grid computing enabling ubiquitous health monitoring of soldiers in a battlefield setting.

consistency check has to be performed on the raw data provided by the failure-prone inexpensive sensors. Secondly, there is a need to prioritize the transmission of huge amount of preprocessed vital sign data from multiple body sites on a patient and from multiple patients (say, in a retirement home or hospital setting) in terms of their degree of importance for diagnosis to avoid network traffic congestion and to maximize reliability. Thirdly, the collected vital signs have to be input into compute-intensive models to derive meaningful physiological parameters of interest. Our research contributions help overcome these challenges as envisioned in Fig. 1.3.

Ubiquitous health monitoring (battlefield setting) In warfare, whether conventional or irregular soldiers inevitably get in harm's way. In such situations, there is a need for real-time in-situ analysis of neuro/physiological parameters under situational context to generate actionable knowledge about soldiers' psychophysiological condition. The knowledge extracted from this analysis can then be exploited 1) to estimate extent of fatigue as well as mental stress and associate it with the current and past activities and/or experiences of the soldier, 2) to assess the adaptability of individuals and the team to high-stress environments, and 3) to infer the overall physical fitness levels and team performance. However, simultaneously executing compute-intensive models for deriving neuro/physiological parameters and algorithms for acquiring context awareness in real time requires computing capabilities that go beyond those of an individual sensor nodes and/or hand-helds requiring solutions such as ours for harnessing the collective computational capabilities of a mobile computing grid as depicted in Fig. 1.4.

1.5 Dissertation Organization

This rest of this dissertation is organized as follows.

Chapter 2 reviews related and prior work in the fields of cyber foraging, opportunistic computing, mobile grid computing, and distributed computing in uncertain environments. The proposed approaches are discussed in detail along with their pros and cons. The arguments in favor of our autonomic uncertainty-aware approach are also presented.

Chapter 3 presents Cumulus and details our uncertainty-aware autonomic middleware for mobile grid computing. Details on the mechanisms for self-organization, self-optimization and self-healing (uncertainty awareness) are presented. Results of a thorough evaluation of the resource provisioning engine (which imparts the self-optimization capability) for data-parallel applications on our prototype testbed are also discussed in detail.

Chapter 4 explains the unified data and computing-resource uncertainty management framework for a broader class of task-parallel applications. A generic mobile-application workflow representation scheme is presented followed by mechanisms to capture propagation of uncertainty up the data-processing chain using interval arithmetic. A novel approach to workflow task distribution based on multi-objective optimization is presented and along with details about performance evaluation.

Chapter 5 describes, `Maestro`, a robust, secure, and privacy-preserving mobile grid computing framework for orchestrating the execution of concurrent mobile application workflows. A sub-graph matching technique, called `Dedup`, for task deduplication among workflows is presented followed by a replication-based task scheduling mechanism that imparts robustness, security (self-protection), and privacy (via multiple levels of authorization).

Finally, details of performance evaluation are discussed.

Chapter 6 summarizes our contributions as well as our observations and provides suggestions for future research directions that will push the state of the art in autonomic resource management of mobile computing grids.

Chapter 2
Background and Related Work

Augmentation of mobile devices' capabilities would enable novel *compute-* and *data-intensive* mobile applications spanning across multiple domains. Therefore, in order to realize seamless augmentation, researchers have tried to solve a plethora of research problems in the broad areas of mobile computing and distributed systems. In this chapter, we present an overview of prior and simultaneous research in the fields of cyber foraging, uncertainty-aware resource management, and application workflow management. We also highlight observations that served as motivations to our approach.

2.1 Cyber Foraging

Research efforts towards realizing the vision of cyber foraging can be broadly classified into works in the fields of *mobile cloud computing*, *opportunistic computing*, and *mobile grid computing.* Presently, wired grid and cloud computing are the most popular approaches to enable compute-intensive scientific and enterprise data processing predominantly operating on supercomputers and/or on computing and storage clusters [13]. Therefore, many works in mobile cloud computing have primarily focused on offloading expensive (compute and energy-intensive) tasks to *dedicated* and *trusted* computing resources, either situated remotely (in the *cloud* [3–5, 7, 14] or in the wired-grid [15–17]) or proximally (in *cloudlets* [6]) in a transparent manner. However, these approaches are not suitable for enabling data-intensive applications in real time due to prohibitive communication cost and response time, significant energy footprint, and the curse of extreme centralization.

As cyber foraging was meant to convey a whole new pervasive computing paradigm based on the principle of "living off the land" [2], works in the fields of Opportunistic Computing (OC) [18–20] and on mobile grid computing [21, 22] emerged due to the ubiquity and growing computing capabilities of mobile devices. These works have explored the feasibility of leveraging the computing and communication capabilities of other mobile devices *in the field* to enable innovative mobile applications. Recent research in the areas of mobile grids (mobile device clouds) [21, 22] and opportunistic computing [18–20] have explored the potential of code offloading to proximal devices by following two entirely different approaches. Solutions for mobile grids advocate a structured and robust approach to workflow and resource management; whereas OC depends entirely on *direct encounters* and is highly unstructured with little or no performance guarantees. However, the aforementioned works do not address the crucial research challenge of “real-time” concurrent applications management while also taking all of the following concerns into account: robustness, security, and privacy. We would like to point out that the wired grids and clouds can be vital elements of our elastic resource pool and can be exploited for complex offline post-processing of sensor data (stored in remote databases) to profile as well as to extract features and trends whenever required.

2.2 Mobile Grid Computing

Reliable coordination (i.e., service discovery, workload distribution) is a necessity in mobile grid for ensuring QoS in terms of response time and energy consumption in the presence of uncertainties [23, 24]. A simple approach is to employ a centralized coordinator that possesses grid information (e.g., location, communication cost, and resource availability). In [25], service discovery as well as workload distribution are performed by a central coordinator. Similarly, the grid may be divided into sub-clusters and be managed by multiple coordinators (managers). In [26], each cluster has a manager that updates grid information within its own cluster and communicates with other managers for coordination. These two approaches are architecturally simple for coordination of grid resources, but they suffer from the communication bottleneck at the manager and the single point of failure can make the system vulnerable. Our approach to the use of multiple coordinators (arbitrators) is not similar to the one by Barbosa et al. [26] as we consider coordination to be only an additional duty for some of the grid resources and do not maintain strict cluster boundaries.

Optimal job allocation on mobile grids is a significant challenge under uncertainties.

Table 2.1: Summary of related work in mobile grid computing.

Focus	Chu et al. [21]	Giurgiu et al. [14]	Chun et al. [3]	Darby et al. [27]	Costa et al. [28]	Huang et al. [29]	Lima et al. [22]	Jiang et al. [30]	Liao et al. [31]	OURS
Infrastructure										
Mobile					✓		✓	✓	✓	
Static						✓				
Hybrid	✓	✓	✓	✓						✓
Concern										
Energy			✓			✓				✓
Connectivity				✓	✓					✓
Uncertainty				✓						✓
Contribution										
Protocol				✓			✓		✓	
Middleware	✓	✓	✓		✓	✓		✓		✓

Chu and Humphrey [21] proposed mobile OGSI.NET, a framework that allows applications to spread tasks in a purely *mobile grid* by creating application modules at multiple nodes and executing tasks in the most appropriate location. However, they focused on making a reliable framework to distribute tasks to multiple devices, but did not consider the problem of optimizing resources utilization for energy efficiency. Lima et al. [22] presented a middleware architecture called MoGrid, which distributes task in a mobile grid based on a peer-to-peer discovery protocol. Their discovery protocol coordinates the distribution of grid tasks by exchanging messages between a node and a collaborator. Thus, the framework allows ad hoc resources to enter or leave the mobile grid and to have the ability to utilize efficiently the available resources. Ghosh et al. [32] proposed an optimal job-allocation scheme for mobile-grid computing by drawing upon the Nash Bargaining solution to maximize grid revenue from the view point of the user. Giurgiu et al. [14] proposed a middleware that optimizes job distribution for latency, data transferred, and cost between the mobile devices and a remote servers. However, these job-allocation schemes are employed based on the current/known resources available, making the solution vulnerable to the changing environment (uncertainties). In Table 2.1, we summarize previous work in mobile grid computing and position our uncertainty-aware resource provisioning framework for real-time in-the-field data processing in mobile computing grids.

2.3 Uncertainty-aware Resource Management

Mobile-grid computing requires robust communication schemes to ensure QoS in the face of inherent uncertainty in the mobile computing infrastructure, in terms of network connectivity, availability, and high susceptibility to hardware failures. Darby et al. [27] proposed the idea of creating a peer-to-peer checkpointing agreement (entrusting duplicates of critical application data and status with trusted peers) for guaranteeing QoS in mobile grids. Costa et al. [28] presented a middleware focusing on collecting data in wireless sensor network to send and receive messages in order to coordinate distributed computing resources on a mobile grid for the application where sensing and acting devices are crucial. Huang et al. [29] proposed an energy-efficient middleware supporting multimedia services to prolong lifetime of mobile grid by transferring computational load from the underpowered mobile devices to the more powerful machines in the grid. However, these communication and coordination solutions for mobile grids are not well integrated with the heterogeneous computing and storage capabilities of electronic devices in the field.

Data and computing-resource uncertainty, if unchecked, may propagate up the data-processing chain and have an adverse effect on the relevance of the generated result (level of accuracy and timeliness). Prior work in the field of data-uncertainty management [33–35] and on computing-resource-uncertainty management [27, 36] identifies and captures the limitations arising out of either imperfect/partial data or insufficient computing resources to provide not-so-accurate yet meaningful results within specified deadlines. However, these works focus on data- or resource-uncertainty management *in isolation* to prevent those from adversely affecting the result. However, uncertainty-aware computing in the mobile sensing and computing domain requires a *unified approach* (for both data as well as resource management) as it is affected by uncertainties arising out of both sources.

2.4 Workflow Management

Concurrent workflows management has been studied before in the context of wired-grid computing. In [37–39], different strategies for scheduling multiple workflows were investigated – namely, sequential (i.e., one after the other), by interleaving tasks of the different workflows in a round robin manner, and by merging the different workflows into one. Independent of the strategy chosen, the workflow tasks are allocated using level-based [38], list-based [37], duplication-based [40], or clustering-based heuristics [39]. In all these heuristics, all the tasks of a workflow are scheduled at the same time with the option of filling the "gaps" (schedule holes due to high communication cost between tasks) for efficient resource utilization.

However, none of the existing solutions can be adopted for workflow management mobile device clouds as they do not factor in task deduplication (for efficiency), reactive self-healing and proactive self-protection (for failure handling), security (from malicious resources), and privacy, which are primary concerns in mobile grid computing. Even though duplication-based scheduling provides some level of redundancy, it treats only fork tasks (ones with multiple successor tasks) as critical and does not protect other tasks that may be critical in the context of the application (or annotated by the developer or user as one). All the aforementioned shortcomings serve as a motivation for our clean-slate design.

In summary, prior research efforts, have aimed at integrating mobile devices into the wired-grid and cloud computing infrastructure mainly as service requesters. In contrast, we exploit mobile devices as service providers, handle concurrent mobile application workflows, and address uncertainty-aware resource management for ensuring application QoS even in highly dynamic and unpredictable environments. To the best of our knowledge, ours is the first work to explore deduplication and scheduling of tasks belonging to concurrent real-time mobile application workflows on mobile grids.

Chapter 3
Autonomic Middleware for Mobile Grid Computing

3.1 Overview

This chapter presents a resource provisioning framework for organizing the heterogeneous sensing, computing, and communication capabilities of static and mobile devices in the vicinity in order to form the Cumulus – a hybrid static/mobile computing grid. Resource provisioning in Cumulus is a challenging problem due to inherent *uncertainty* in terms of network connectivity and device availability. This uncertainty can be attributed to unpredictable node mobility, varying rate of battery drain, susceptibility to hardware failures, and lack of a priori knowledge about the application performance on different mobile hardware and software platforms. In order to address the research challenges associated with reliable hybrid grid coordination and application performance (in terms of response time) under uncertainty, we impart our proposed resource provisioning framework with autonomic capabilities, namely, *self-organization, self-optimization,* and *self-healing.* Our contributions are geared towards imparting autonomic capabilities to the resource management framework. Mechanisms for service discovery and workload management will impart the self-organization capability. The energy-aware resource provisioning engine will impart self-optimization while the uncertainty handling mechanism will bestow the self-healing capability.

3.2 Cumulus

The entities of a Cumulus may at any time play one or more of the following three *logical roles* as shown in Fig. 3.1: i) *service requester,* which places requests for workloads that require additional data and/or computing resources from other devices, ii) *service provider*

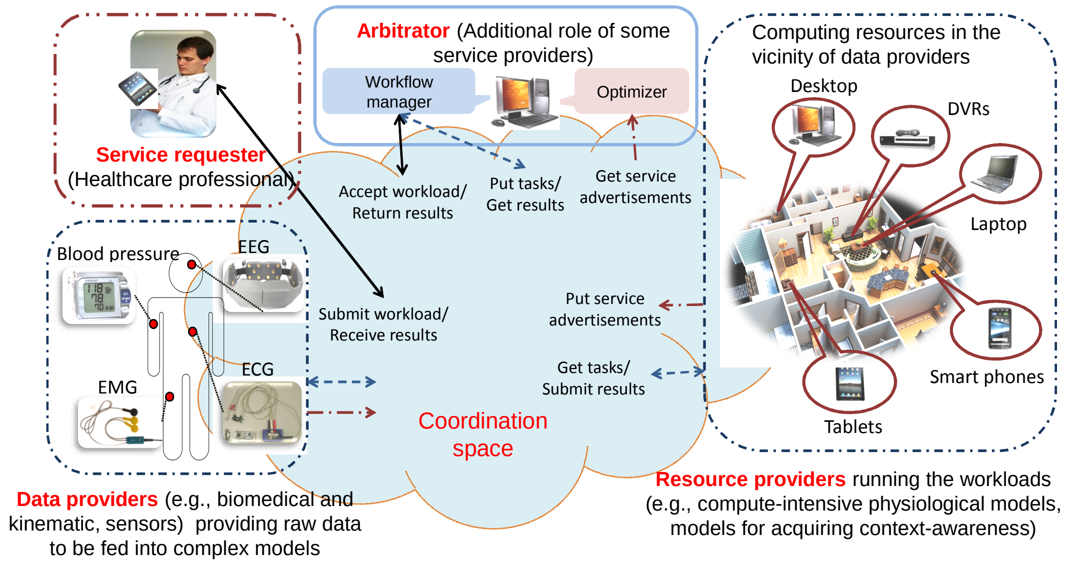

Service requester
(Healthcare professional)
Arbitrator (Additional role of some service providers)
Workflow manager
Optimizer
Computing resources in the vicinity of data providers
Desktop
DVRs
Laptop
Smart phones
Tablets
Accept workload/ Return results
Put tasks/ Get results
Get service advertisements
Blood pressure
EEG
ECG
EMG
Submit workload/ Receive results
Put service advertisements
Get tasks/ Submit results
Coordination space
Data providers (e.g., biomedical and kinematic, sensors) providing raw data to be fed into complex models
Resource providers running the workloads (e.g., compute-intensive physiological models, models for acquiring context-awareness)

Figure 3.1: Overview of the logical roles in a Cumulus and how it can power an ubiquitous healthcare and monitoring application.

(SP), which can be a *data provider (DP), resource provider (RP),* or both, and iii) *arbitrator* (also typically known as broker), which processes the requests from the requesters, determines the set of service providers that will provide or process data, and distributes the workload tasks among them. Data providers provide scalar or multimedia data while resource providers lend their computational (CPU cycles), storage (volatile and non-volatile memory), and communication (i.e., network interface capacity) resources for processing data. The arbitrator – an additional role played by some of the service providers – is aided by a novel *uncertainty- and energy-aware resource allocation engine,* which will distribute the workload tasks optimally among the service providers. The arbitrators are in charge of handling concurrent service requests as well as of orchestrating the execution of mobile applications on SPs in a robust, secure, and privacy-preserving manner.

This role-based architecture enables easy management and does not suffer from the problem of extreme centralization. The self-organization capability (for handling service discovery and service request arrivals as well as for task distribution and management) is imparted by the role-based architectural framework. It also facilitates interactions among the mobile entities for coordination and seamless switching among the three logical roles, namely, *service requester, service provider,* and *arbitrator.*

Service discovery: Service discovery at the arbitrators is achieved through voluntary service advertisements from the service providers. Service advertisements will include information about the current position, amount of computing (γ_n^{cpu}, in terms of normalized CPU cycles), memory (γ_n^{mem} [Bytes]), and communication (γ_n^{net} [bps]) resources, the start (t_n^{in}) and end (t_n^{out}) times of the availability of those resources, and the available battery capacity (e_n^{adv} [Wh]) at each service provider n. The arbitrator is aware of the instantaneous power drawn by the workload tasks of a specific application when running on a specific class of CPU and memory (together given by c_n^{comp} [W]) as well as network (c_n^{net} [W]) resources at each service provider as the information about the different types of devices is known in advance. The arbitrators use the information from service advertisements of the N computing devices to derive the following: $\mathbf{R} = \{r_{mn}\}_{N \times N}$ [m], which conveys the distance r_{mn} between devices m and n, $\overline{S} = \{s_n\}_{1 \times N}$, where $s_n \in \{1, 0\}$, which conveys whether n is a resource provider or not, and $\overline{D} = \{d_n\}_{1 \times N}$, where $d_n \in \{1, 0\}$, which conveys whether n is a data provider or not.

We advocate the use of a distributed self-election mechanism for assigning the appropriate number of arbitrators. Distributed arbitrator self-election is a non-trivial challenge as too many arbitrators can cause network congestion with excessive control overhead (i.e., communication messages between arbitrators and data/service providers) while too few can compromise robustness when arbitrators fail. We leverage our prior experience in distributed adaptive sampling in wireless sensor networks [41] where we were faced with the problem of selective representation to reduce communication overhead while still minimizing the error in reconstruction of the underlying phenomenon. Our self-election mechanism works as follows: each service provider will determine the potential size of its resource pool, i.e., the number of *service advertisements* it has received. Then, all the service providers advertise this number and determine their *rank* in their neighborhood in terms of *influence* (potential size of their resource pool). The service providers use a pre-determined rank threshold (depending on the network size and density) to elect themselves as arbitrators.

Workload management: Each arbitrator is composed of two components, namely, *workload manager* and *scheduler/optimizer,* as shown in the top of Fig. 3.1. The workload manager (also called *master*) tracks workload requests, allocates workload tasks among service providers, and aggregates results. The optimizer identifies the number of service providers (also called *workers*) available for the requested duration and determines the optimal distribution of workload tasks among them. The optimizer shares the workload submitted by the data providers among the available service providers based on one of several possible policies.

The different tasks of a workload may be distributed among the available service providers based on a policy that aims at minimizing the battery drain. This can be achieved through minimization of computational load on each individual service provider by exploiting parallelism while incurring a very low communication cost. Another policy may just place emphasis on response time without considering battery drain. Our framework applies to applications exhibiting *data parallelism* (in which data is distributed across different parallel computing nodes that perform the same task) as well as to applications exhibiting *task parallelism* (in which parallel computing nodes may perform different tasks on the same or different data).

3.3 Resource Allocation Engine

Here, we explain our uncertainty-aware resource allocation engine (an optimization problem corresponding to one of the aforementioned policies) for hybrid grids in detail. In the following, we explain the sequence of events happening at one of the arbitrators while similar events happen simultaneously at the other arbitrators in the computing grid. When a service requester needs additional data or computing resources, it submits a service request to the nearest arbitrator and also specifies δ^{max} [s], the maximum duration for which it is ready to wait for a service response. The arbitrator extracts the following information based on the service advertisements: the devices' (service providers') capability, $\overline{\Gamma}^{x} = \{\gamma_n^x\}_{1\times N}$, where $x = cpu, mem, net$; the associated costs, $\overline{C}^{comp} = \{c_n^{comp}\}_{1\times N}$ and $\overline{C}^{net} = \{c_n^{net}\}_{1\times N}$; the devices' availability, $\overline{T}^{in} = \{t_n^{in}\}_{1\times N}$ and $\overline{T}^{out} = \{t_n^{out}\}_{1\times N}$; and their battery status $\overline{E}^{adv} = \{e_n^{adv}\}_{1\times N}$. The variables that the optimization problem has to find are,

$$\textbf{Find}: \ \mathbf{A}, \overline{U}, \overline{\Delta}^{d}, \overline{\Delta}^{s}. \tag{3.1}$$

Here, $\overline{U} = \{u_n\}_{1\times N}$ with $u_n \in \{1, 0\}$ conveys whether a resource provider n is used for computing or not, $\overline{\Delta}^d = \{\delta_n^d\}_{1\times N}$ [s] conveys the duration for which the services of each service provider will be used for data collection, and $\overline{\Delta}^s = \{\delta_n^s\}_{1\times N}$ [s] conveys the duration for which the resources of each service provider will be used for computation (*cpu*, *mem*, and *net*) and/or for multi-hop communication (*net*) as a relay node. In this formulation, the objective of the optimization problem, given by (3.2) and (3.3), is *maximization of minimum residual battery capacity* at all the service providers, max $\min_n e_n^{res}$ [Wh], while ensuring that the service response is delivered within δ^{max}. This objective maximizes the lifetime of every single service provider and, thus, maintains the heterogeneity of the resource pool for longer periods. The set of service providers and the duration for which each of their capabilities are availed will be determined by considering the trade-offs among the cost (in terms of battery drain) e_n^{data} [Wh] (3.4) for transferring the data locally from data providers to the resource providers, the computational cost e_n^{comp} [Wh] (3.5) for availing the computational capabilities of the resource providers for servicing the request and for aggregating and generating the final response.

$$\textbf{Maximize}: \ \min_n e_n^{res}, \tag{3.2}$$

$$where, \ e_n^{res} = e_n^{adv} - (e_n^{data} + e_n^{comp}); \tag{3.3}$$

$$e_n^{data} = \frac{\delta_n^d}{3600} \cdot c_n^{net}; \tag{3.4}$$

$$e_n^{comp} = u_n \cdot \frac{\delta_n^s}{3600} \cdot c_n^{comp}. \tag{3.5}$$

In (3.3), $e_n^{data} + e_n^{comp}$ is the amount of battery capacity drained at each service provider n. δ_n^d for a service provider n depends on the amount of data it has to transmit (ω [Bytes] as a data provider) or aggregate ($\omega \cdot \sum_{i=1}^N a_{in}$ [Bytes] as a resource provider), and the availed communication capability, given by,

$$\delta_n^d = \begin{cases} f(\omega, \gamma_n^{net}), & \text{if } u_n = 0 \\ f(\omega \cdot \sum_{i=1}^N a_{in}), & \text{if } u_n = 1. \end{cases} \tag{3.6}$$

For simplicity, ω is considered to be the problem size of a trivial task and each data provider provides the same amount of data. However, this is easily generalizable to a case where each data provider provides a different amount of data, in which case the problem sizes of each trivial task will be different.

Matrix $\mathbf{A} = \{a_{ij}\}_{N \times N}$ conveys the associativity of data provider i with service provider j, which is determined by the arbitrator. Function f monotonically increases as the amount of data to be transmitted or received increases. δ_n^s for a service provider n depends on the amount of data it has to process and the availed computing capabilities specified by γ_n^{cpu} and γ_n^{mem}, given by,

$$\delta_n^s = g(\gamma_n^{cpu}, \gamma_n^{mem}, \omega \cdot \sum_{i=1}^{N} a_{in}). \tag{3.7}$$

Function g monotonically increases with the amount of data to be processed. The constraints to the optimization problem are,

$$\begin{aligned}
&\underline{\textbf{\textit{Constraints:}}}\ \forall n = 1 \ldots N, \\
&s_n \geq u_n; && (3.8)\\
&0 \leq \delta_n^d, \delta_n^s; && (3.9)\\
&\delta_n^s \leq \min\{t_n^{out}, t^{now} + \delta^{max}\} + \\
&\quad - \max\{t^{now} + \delta_n^d, t_n^{in}\}; && (3.10)\\
&\frac{\delta_n^d}{3600} \cdot c_n^{net} + u_n \cdot \frac{\delta_n^s}{3600} \cdot c_n^{comp} \leq e_n^{adv}. && (3.11)
\end{aligned}$$

Constraint (3.8) ensures that only a resource provider is chosen to perform the computing. Constraints (3.9) and (3.10) ensure that the consumer's deadline for service response is met while also utilizing a service provider only for the duration for which its services are advertised to be available. Constraint (3.11) ensures that the advertised battery capacity is not exceeded.

3.3.1 Uncertainty Awareness

The resource allocation engine of our proposed in-situ data processing system is capable of handling uncertainties in the highly dynamic hybrid heterogeneous computing environment. We identify the different sources of uncertainties and bestow the resource allocation engine with the desired properties to guarantee application QoS (in terms of response time) even under those uncertainties.

Sources of uncertainty: *Inaccurate estimation of the availability (duration) of service provider* is a major source of uncertainty that results in a large number of incomplete workload task migrations. The duration of availability specified in the service advertisements is based on the battery drain estimates and may not accurately reflect the duration for which the service provider will be associated with the arbitrator. One or more of the service providers may lose network connectivity to the arbitrator or go offline.

Inaccuracy in the estimation of task completion times – function $g()$ in the aforementioned optimization problem – is one of the sources of uncertainties that affect application QoS. This is especially true when the behavior of a workload task (execution time and resource utilization) is not known in advance at the arbitrator. The uncertainty can be reduced to a certain extent in our application scenario by profiling the behavior of the workload in advance. *However, some models exhibit radically different behaviors depending on the type of inputs (e.g., sorted/unsorted, dense/sparse).*

Uncertainty in workload completion within the response-time bound arises when a service provider is experiencing an unexpected *increase in the rate of battery drain* (and runs out of energy) due to any of the additional critical operations that it may be performing at the same time as the workload task. The optimization problem may also *over- or under-provision* computing resources due to an inaccurate estimate of task completion time. The former would result in unnecessary wastage of energy (battery drain) while the latter would result in violation of QoS (in terms of response time). While the arbitrator can be made aware of any problems at the service provider using feedback, handling situations such as loss of network connectivity and hardware failures is a challenging task.

Uncertainty-aware self-organization: We advocate the use of multiple arbitrators to avoid a single point of failure. The arbitrators play a very important role in handling uncertainties caused by the unavailability of service providers or by the inaccuracy of task-completion-time estimates. In order to ensure that the unavailability of an arbitrator (due to poor connectivity or hardware failure) does not lead to the failure of the entire system, each arbitrator shares with all of its active data and service providers a *list of alternate arbitrators* – referred to as *proxies* – ranked according to their proximity (primary key) and physical addresses (secondary key). In case of an arbitrator failure, the service providers collaborate with the pre-specified proxy until the end of all active workload tasks. The arbitrators also share their current state information – namely, data providers and service requests that are currently being served as well as the list of service providers currently employed – with their proxies to handle any unexpected failures. This approach has been exploited previously in grid computing when federating different grids, each with its own resource broker [42,43]. In addition to robustness and self-organization, the use of a robust underlying overlay network of arbitrators/proxies can support policies when different arbitrators have access to different resource providers (heterogeneity across different resource pools). In such a case, existing techniques and policies could be used to optimally match provider with requirements, as existing work has already studied in the context of grid computing.

In order to impart the uncertainty-aware self-organization capability to the proposed resource-allocation framework, we designed mechanisms that help the arbitrator extract the following long-term statistics from the underlying resource pool: the average arrival (joining) rate of service providers ($\widetilde{W}$), the average service provider availability duration ($\widetilde{T}$), and the average number of service providers associated with the arbitrator at any point in time ($\widetilde{N}$). The relationship among these three long-term statistics is given by Little's law [44], $\widetilde{N} = \widetilde{W} \cdot \widetilde{T}$. The arbitrators update continuously these statistics and share at least two of the three aforementioned averages with its successors if and when an arbitrator's handoff happens. Knowledge of these average statistics helps the arbitrators assess the *churn rate* of service providers. Churn rate is a measure of the number of service providers moving into or out of an arbitrator's resource pool over a specific period of time. Note that the arbitrators need not extract or be aware of the underlying probability distribution of service provider arrivals or of availability durations. The long-term averages, which are easy to acquire and maintain, are sufficient.

Churn rate of service providers will be different in different geographic location. For example, the churn rate of service providers at a shopping mall is far greater than the one at a coffee shop. Also, at a particular location (say, the coffee shop), the churn rate can vary over time (e.g., depending on the time of the day). When the churn rate of service providers is high, i.e., the average duration of service providers availability is low, the percentage of potentially costly migrated workload tasks will be high if the resource-allocation engine does not possess uncertainty awareness. When the long-term average of availability duration is not taken into account at the arbitrator and when the durations advertised by the service providers are used as constraints in the optimization problem (presented in the previous section), it results in a mismatch between the ground reality and the optimization at the arbitrator. However, our framework with uncertainty awareness achieves a smooth degradation (if any) in QoS (because of the small number of task migrations) when churn rate increases as it effectively exploits the knowledge gathered over time and/or acquired from its predecessors.

3.3.2 Uncertainty Handling

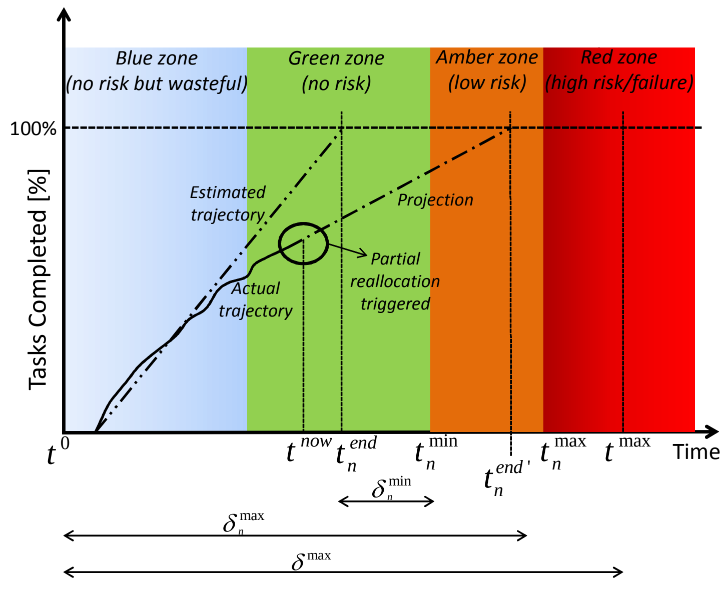
To ensure application QoS (in terms of response time) under all of the aforementioned circumstances, we introduce the novel idea of *application waypoints*, at which the service providers report to the arbitrator with intermediate results and their progress. Waypoints enable the arbitrators to estimate the *residual tasks' execution time* for each service provider. If an arbitrator does not receive any waypoints from a service provider, it marks that service providers as failed after a timeout and assigns the incomplete tasks to one or more backup service providers. If an arbitrator receives waypoints from a service provider at a lower rate than the expected (from offline application profiling), it has to determine *when* and *how* to intervene, i.e., when to relieve the slow service provider of some tasks and which service providers to use as backup. The details of our proposed reactive measurement-based self-healing mechanism for heterogeneous mobile computing grids follow.

Application waypoints: Once the arbitrator assigns sets of tasks to the different service providers, it continuously tracks their progress using application waypoints. In data-parallel applications, the completion of every single workload task is used as a way-point *and* as an opportunity to collect intermediate results at the result aggregator (in some cases, the arbitrator itself). Without any loss in generality, let us assume that the tasks in the data-parallel application are homogeneous. Let v_n be the *rate of application waypoints* (number of tasks completed in unit time) from a service provider n, which is estimated using information obtained during application profiling and during resource allocation. The number of assigned tasks, the time taken for computation δ_n^s, and the time taken for communication δ_s^d at the different service providers are taken into account in this estimation procedure. Therefore, the arbitrator has an initial estimate of the completion time t_n^{end} at n.

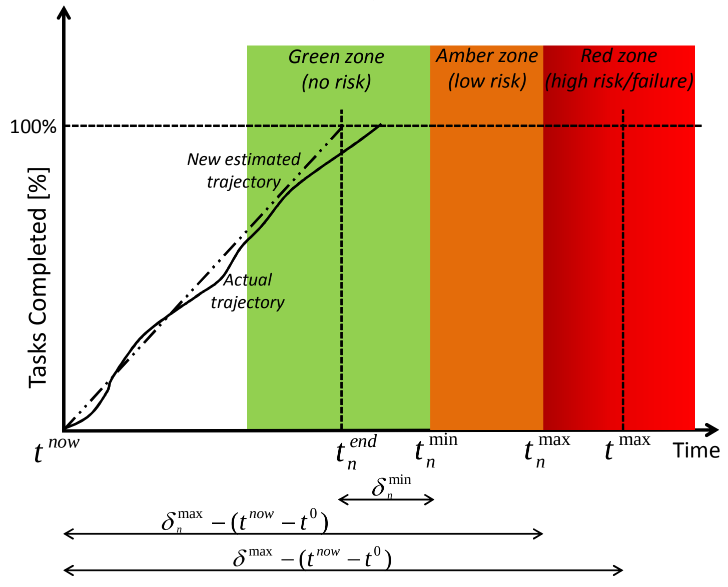
As shown in Fig. 3.2(a), every service provider has a deadline $t^0+\delta_n^{max}$ for the completion of all the tasks that it has been allocated. Here, $\delta_n^{max} < \delta^{max}$ in order to give a margin for result aggregation. As the actual rate of waypoints v_n' from the service provider n will be different from the initial estimate v_n, the arbitrator should be able to absorb this variation and react appropriately depending on where the projected completion time $t_n^{end'}$ of all the allocated tasks lies in relation to the individual service provider deadline t_n^{max}. We identify *four end zones*, namely, Blue, Green, Amber, and Red, as depicted in Fig. 3.2: the arbitrator's reaction will depend on the zone in which the projected completion time $t_n^{end'}$ falls in. This projection is straightforward as the arbitrator is aware of the number of tasks allocated to each service provider and of the corresponding v estimates.

Red zone (high risk/failure): When $v_n' \ll v_n$ and $t_n^{end'}$ falls in the Red zone (beyond t_n^{max}), it means that the service provider n is completing the allocated tasks very slowly or the waypoints are not received at the arbitrator. The arbitrator determines that the particular service provider is unable to complete all the tasks that have been allocated to it within t_n^{max} and reallocates "all" the residual tasks to backup service providers. There is, however, a possibility that this scenario arises due to a very poor estimation of v_n, in which case the profile needs to be updated. This will be done only after ensuring repeatability of this issue.

Amber zone (low risk): When $v_n' < v_n$ and $t_n^{end'}$ falls in the Amber zone (as shown in Fig. 3.2(a)), it means that the service provider n is completing the allocated tasks slower



(a)



(b)

Figure 3.2: Illustration of the use of *application waypoints* to report progress of workload task completion at a service provider n. (a) shows the estimated trajectory of task completion and the relative position of the estimated completion time t_n^{end} with respect to the four end zones Blue, Green, Amber, and Red. The projection at time t^{now} falls in the Amber zone triggering a partial reallocation of tasks from this troubled service provider; (b) shows the new estimated and the actual trajectory of task completion at the troubled service provider after it has been relieved of some tasks. The service provider is able to complete its allocated tasks with the service providers' deadline t_n^{max}. If the projection had fallen in the Red zone, it would have triggered a full reallocation, i.e., all the incomplete tasks at the service provider.

than what was estimated during profiling. The Amber zone is bounded by t_n^{min} and t_n^{max}. Even though the projected completion time falls within the service provider deadline, the uncertainty is deemed unacceptable by the arbitrator, which reallocates a “fraction” of the residual tasks to backup service providers. The arbitrator wants the lower bound of the Amber zone t_n^{min} to be as close as possible to the deadline t_n^{max} so to absorb the effect of acceptable variation in v'_n from v_n. On the other hand, the arbitrator also takes into account the capabilities/limitations of the backup resource pool to complete the reallocated tasks within the deadline. Therefore, δ_n^{min} – which determines t_n^{min} – is a function of multiple factors, i.e.,

$$\delta_n^{min} = \Phi(v_n, v'_n, \widetilde{N}, \widetilde{T}, t_n^{end}, t_n^{max}). \tag{3.12}$$

When the average size of the resource pool $\widetilde{N}$ and the average availability duration $\widetilde{T}$ of the resource pool are large, δ_n^{min} is large (Amber zone is smaller), i.e., the arbitrator can wait longer before intervening as the resources at its disposal are capable (in terms of execution time) of quickly finishing the incomplete tasks after a reallocation is triggered. On the contrary, when the resources at the disposal of the arbitrator are limited, the Amber zone is larger as the arbitrator and the resource pool needs more time to react after a reallocation is triggered.

Green zone (no risk): When $t_n^{end'}$ falls in the Green zone, it means that the current operating conditions at the service provider is not significantly different from the one during profiling. This rate of progress is close to the desired trajectory for workload task completion and the arbitrator does not react in such a scenario. In other words, the level of uncertainty is acceptable to still complete all the allocated tasks within t_{max}^n.

Blue zone (no risk but wasteful): When $v'_n > v_n$ and $t_n^{end'}$ falls in the Blue zone, it means that the service provider n is completing the allocated tasks faster than what was estimated during profiling. This may happen when the profiling is done under non-ideal conditions. The arbitrator uses this opportunity to tune the application profile for the specific hardware and software platform on the service provider n. In some cases, the arbitrator may deem it undesirable if additional energy (battery drain) is being used up for an unnecessary earlier completion of the task and may reallocate the tasks to other service providers to avoid energy wastage. The upper bound for Blue zone t_n^{min} is a tunable parameter that the application developer can set. In our prototype, we set $t_n^{min} = t_n^{end} - \delta_n^{min}$.

While the dynamics are presented at the service provider level in Fig. 3.2, the same phenomenon can be explained at the arbitrator level by combining waypoint information from all the service providers. Application waypoints could be also seen as indicators of progress similarly to the approach described in [45], which provided a general-purpose API and runtime system to implement progress and performance indicators of individual high-performance computing applications. Similar to the waypoints discussed in this chapter, the main purpose of the indicators was improving scheduling policies based on dynamic load-balancing techniques and self-tuning in run time. The concepts and interfaces proposed in [45] can be extended to implement waypoints also at the mobile-OS layers and not just at the application layer as it is done now. Such a general-purpose API, in conjunction with checkpointing [27], will also enable extension of the same principles for self-healing in task-parallel applications with waypoints built "inside" the tasks.

Reallocation strategy: In response to a *partial* or *full* reallocation trigger raised in the Amber or Red zone, the arbitrator provisions additional computing resources and reallocates the workload task(s) so to ensure that the workload is completed within the specified deadline. In contrast to the energy-aware optimization approach for initial task allocation, the reallocation strategy is a heuristic aimed at minimizing the execution time of residual tasks. Firstly, this shift from optimization to heuristic is motivated by the need to react fast. Secondly, the need to reprovision resources and to reallocate tasks arises due to inaccuracies in the models (for task execution time, service provider availability) used in the optimization approach. Therefore, during recovery, while the models are being tuned for future use, a fast heuristic approach is employed. *We propose a best-fit heuristic to reallocate the incomplete tasks to the backup service providers.*

Backup resource pool: The backup pool is first created and the tasks are allocated to these service providers in pool using the best-fit allocation principle [46]. The eligibility criteria for a service provider to be part of the backup pool is $t_n^{start} + \widetilde{T} > t^{max}$. The eligible service providers are then arranged in the decreasing order of i) the rate of application waypoints (primary key) and ii) residual availability duration (secondary key). This ordering gives preference to faster service providers with longer average availability duration.

Task allocation: The fraction of tasks that have to be reallocated depends on the zone in which the projected completion time lies. When $t_n^{end'}$ is in the Red zone, "all" the incomplete tasks at the service provider n are reallocated; whereas when $t_n^{end'}$ is in the Amber zone, a "fraction" of tasks at the service provider n are reallocated. This fraction is a tunable parameter that can either be set to a predefined value or be set on the fly based on the capability of the backup pool (size as well as heterogeneity). In our prototype, we set the fraction to be the *nearest quartile of the number of incomplete tasks.*

The fit criteria in the best-fit heuristic is maximization of the minimum residual idle time across service providers after the allocation of all incomplete tasks. The philosophy behind this reallocation approach is load balancing in the backup resource pool so to minimize makespan while at the same time using as few resources as possible. This reallocation does not incur significant computational overhead as the size of the allocation problem is very small. If N is the number of service providers in the backup pool and M is the number of tasks to be reallocated, then the time complexity is given by $\mathcal{O}(M \cdot \log M + M \cdot N)$, where the first component is due to the sorting procedure. Also, note that our heuristic is different from the best-fit-decreasing algorithm for bin packing as we order the bins (service providers) and not the objects (tasks).

3.4 Evaluation

We have implemented a small-scale prototype of the proposed autonomic framework and performed an empirical evaluation. We have also used simulations to show the scalability of the proposed framework beyond ten nodes (the size of our testbed). In the following sections, firstly, we present details about our testbed and our experiment methodology. Then, we discuss specific experiment scenarios and the results that demonstrate the uncertainty-aware self-organization, self-optimization, and self-healing properties of our proposed framework.

Table 3.1: Heterogeneity of computing devices in the testbed

	Samsung Galaxy Tab	**Motorola Atrix 2**	**Samsung Galaxy S**	**LG Optimus**	**HTC Desire HD**	**Dell Netbook**	**Dell Inspiron Laptop**
CPU	1 GHz Dual-core ARM	1 GHz Dual-core ARM	1 GHz ARM	600 MHz ARM	1 GHz ARM	1 GHz Atom	2 GHz Dual-core Intel
Memory (RAM)	1 GB	1 GB	512 MB	512 MB	786 MB	1 GB	2 GB
Network	802.11 b/g/n, Bluetooth	2G, 3G, 4G, 802.11 b/g/n, Bluetooth	2G, 3G, 802.11 b/g/n, Bluetooth	3G, 802.11 b/g, Bluetooth	2G, 3G, 802.11 b/g/n, Bluetooth	802.11 b/g/n	802.11 b/g, Bluetooth
Battery capacity	7000 mAh	1740 mAh	1500 mAh	1500 mAh	1400 mAh	4500 mAh	5000 mAh
Battery voltage	4 V	3.8 V	3.8 V	3.7 V	3.8 V	11.1 V	11.1 V

3.4.1 Testbed and Experiment Methodology

Heterogeneous devices: The testbed consists of Android- and Linux-based mobile devices with heterogeneous capabilities (summarized in Table 3.1). In our prototype, communications among the master and workers as well as among the optimizer and workers happen over *Comet Space* [47], a scalable peer-to-peer content-based coordination space developed at the Cloud and Autonomic Computing Center, Rutgers University. The messages in this coordination space (or information space) are constructed in the form of tuples (XML strings).

The workload: The mobile application that we used for our experiments is distributed object recognition. In this application, the service requester (which is also the data provider) submits an image of any object that needs to be recognized while also specifying a deadline. The predominant workload in this application is matrix multiplication and the most fundamental workload task is vector multiplication, which is assigned to the different service providers. Distributed object recognition is representative of the wide range of data-parallel applications that our framework can support. Figure 3.3 shows the time taken by the different mobile devices listed in Table 3.1 to complete all the workload tasks when operating in isolation. For near-real-time performance, the delay needs to be in the order of tens of seconds and the data from Fig. 3.3 clearly motivates the need to divide the tasks among service providers in the vicinity for speed up. A demo of our prototype powering a distributed object recognition application in Android-based tablets can be found in the Cyber-Physical

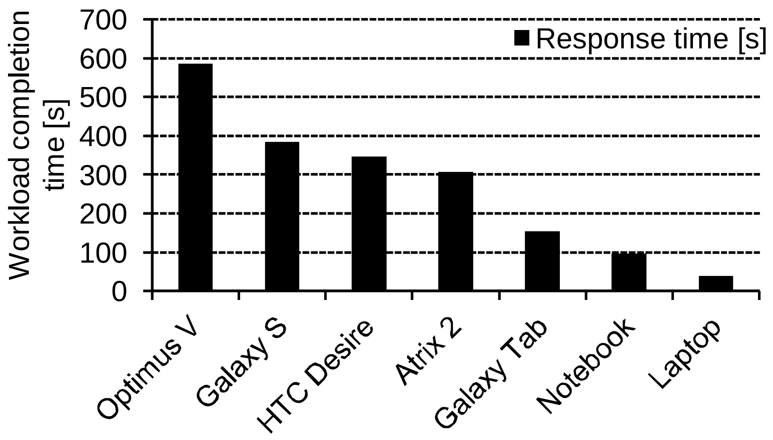


Figure 3.3: Time taken by the individual devices to complete all the workload tasks of the object recognition application.

Systems (CPS) Lab webpage [48].

Application profiling: As the objective of the optimization problem is maximization of minimum residual battery capacity, the amount of battery drain in service providers as a result of running workload tasks needs to be calculated. However, the usage of actual Watt-hour (Wh) values will result in unfair usage of resources in devices with a higher battery capacity. Hence, in order to deal with the heterogeneity of mobile devices with different battery capacities (shown in Table 3.1), in our prototype, the optimization problem uses the residual battery capacity percentage to make allocation decisions. Usage of percentage values instead of actual Wh values ensures fairness in usage of the heterogeneous pool of service providers.

In order to optimize allocation decisions, it is important to get a good estimate of the instantaneous power drawn in the mobile device while running a workload task. This is obtained as follows: we ran all the workload tasks of our object recognition application on the individual mobile devices to determine the instantaneous power drawn (i.e., to determine the current drawn in mA as the voltage drop remains constant) and the total time taken for the workload completion. Figure 3.4 shows the current drawn (negative values as it is current drained from the battery) and the time taken for workload completion when all the 76 tasks of the object recognition application are executed in a Samsung Galaxy Tab. The voltage values did not show significant variability. From these observations, we determined the average time taken to complete one task. This is straightforward as object recognition

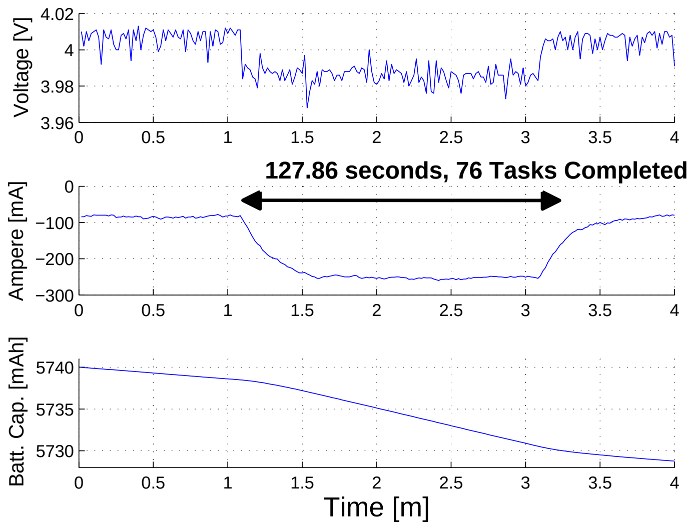


Figure 3.4: Profiling the current drawn [mA] by the workload tasks of the object recognition application running on a Samsung Galaxy Tab.

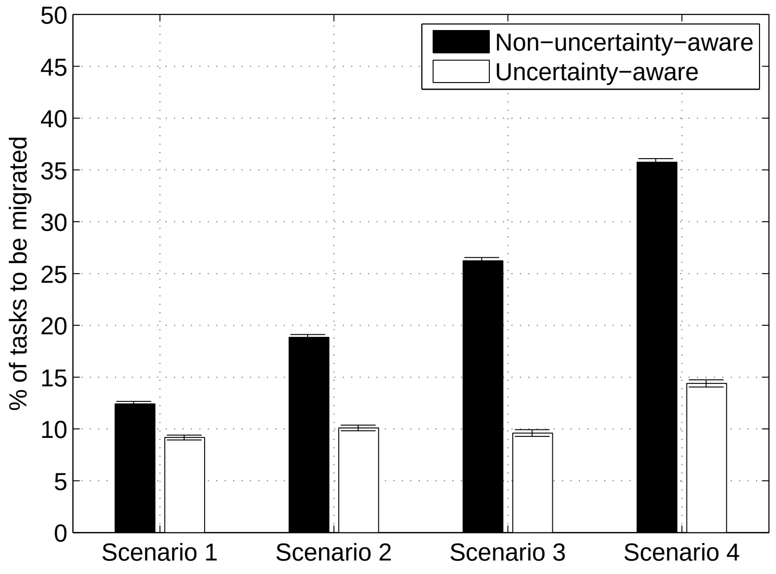
is a data-parallel application whose task (vector multiplication) completion time is not affected by the type of input. In our testbed experiments and simulations, information about task completion time along with the current drawn by the workload tasks, the number of tasks allocated to a service provider, and the present current usage in the service provider helps us calculate the resulting battery drain in Wh. Information about the battery current consumption is readily available in most Android-based devices. However, in the rare case when the battery driver does not make the current consumption information readily available, it can be estimated from the rate of battery drain and the start and end times of the application of interest. Current consumption under idle (or standby) conditions can also be obtained in a similar manner.

3.4.2 Uncertainty-aware Self-organization

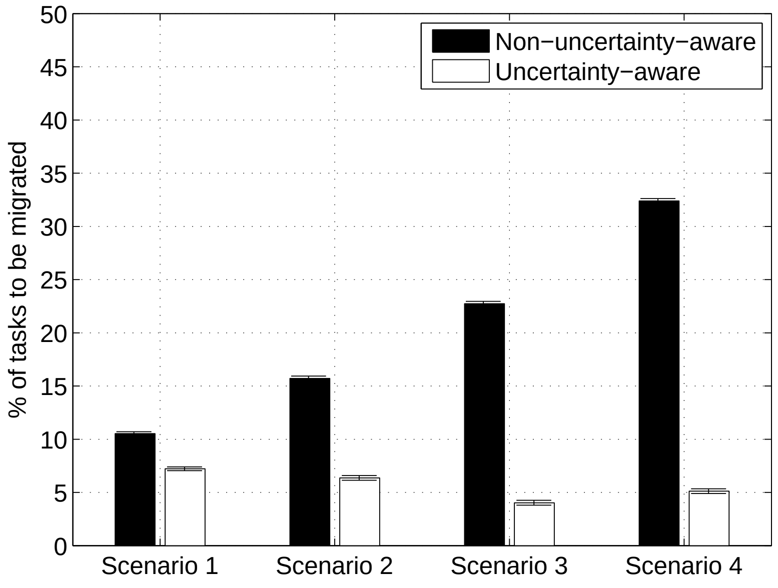
Simulation setup: In order to show the uncertainty-aware self-organization capability of the proposed resource-allocation framework, we performed a simulation to ascertain the gain in terms of reduction in number of workload task migrations that can be achieved by using our framework. We performed simulations under different operational scenarios with different service provider churn rates. In order to achieve different churn rates, we progressively decreased the average time a service provider is associated with an arbitrator. The four scenarios elaborated in the table in Fig. 3.5 represent a progressive increase in the churn rate of the underlying resource pool (with a corresponding decrease in average duration of availability). The number (15 in total) and combination of service providers in the mobile grid, the number of workload tasks, and the deadlines remain the same for all the four scenarios.

We used *percentage of migrated workload tasks* to determine the effectiveness of uncertainty awareness. In order to ensure that the uncertainty awareness capability is not dictated by any particular distribution of service-provider-availability duration, it was picked at random based on i) normal distribution (with mean, $\mu = 180, 150, 120, 90$s and standard deviation, $\sigma = 60$s) and then on ii) Weibull distribution (with scale, $\lambda = 200, 175, 150, 125$ and shape, $k = 4$). Normal distribution is used for its generality while Weibull distribution is the most popular choice amongst statisticians performing reliability (or survivability) analysis [49]. In order to give statistical relevance to our experiments, we performed multiple trials (by picking availability durations from the aforementioned distributions) until we achieved a very small relative confidence interval (less than 10%).

Observations: Figures 3.5(a) and (b) shows how the arbitrator leverages its knowledge of the long-term average of service provider availability in order to reduce the number of workload task migrations. As the churn rate of service providers increases, i.e., the average duration of service providers availability decreases, the percentage of migrated workload tasks increases when we use our resource allocation engine "without" uncertainty awareness. This is because, when the long-term average of availability duration is not taken into account at the arbitrator, the advertised durations (from service providers) are used as constraints in the optimization problem. This leads to a mismatch between the ground reality and the optimization at the arbitrator. However, our framework with uncertainty awareness achieves a smooth degradation (if any) in QoS (because of the small number of task migrations) when churn rate increases as it effectively exploits the knowledge gathered over time and/or acquired from its predecessors. Also, another advantage of uncertainty awareness is that it helps decrease churn rate, especially service provider departures caused by device users opting out of the application due to undesired battery drain. The service providers will not



(a)



(b)

Parameters used in different scenarios.

	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
# Tasks	300	300	300	300
Deadline [s]	240	240	240	240
Availability ($\widetilde{T}$)	180	150	120	90
# Atrix 2	8	8	8	8
# Galaxy S	2	2	2	2
# Optimus V	2	2	2	2
# HTC Desire	3	3	3	3

Figure 3.5: Percentage of migrated workload tasks when the resource allocation engine is uncertainty-aware and otherwise. Effectiveness of uncertainty-awareness when service providers' availability duration follows (a) normal distribution and (b) Weibull distribution.

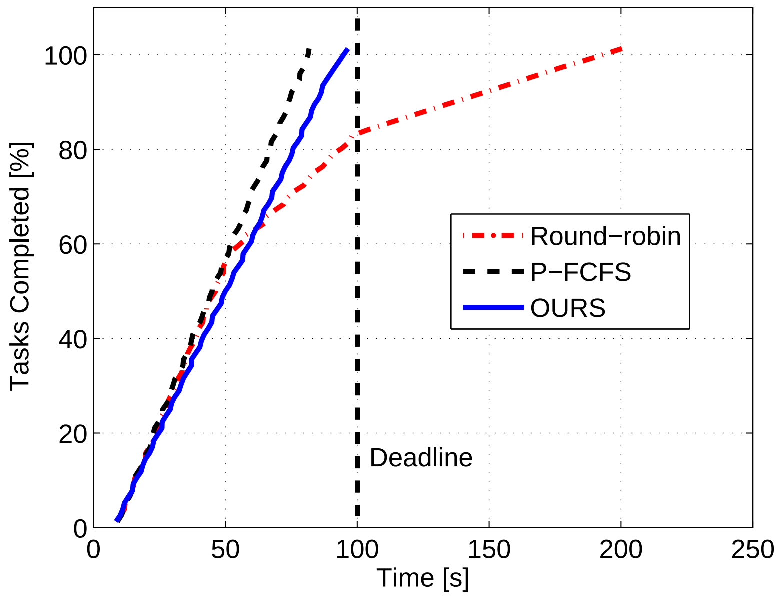
experience undesired and unfair battery drain, especially the more powerful devices, which are usually preferred for strict deadline requirements.

3.4.3 Self-optimization

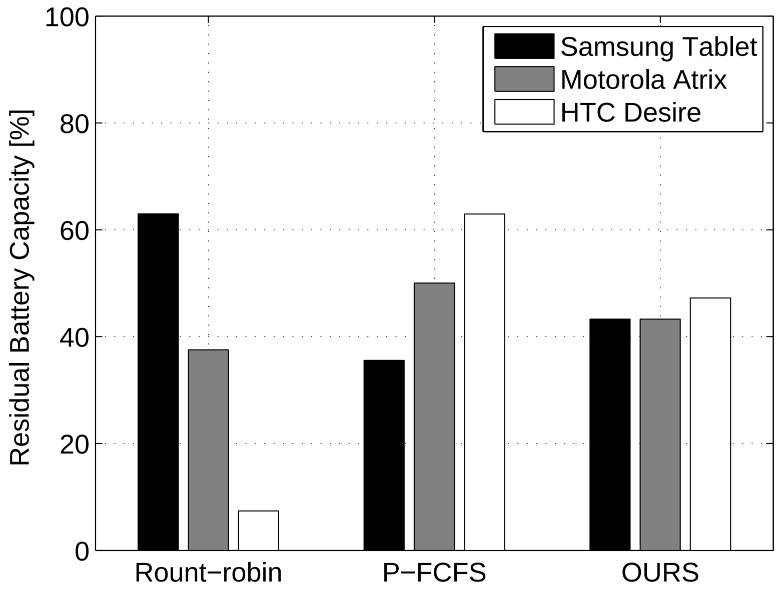
Competing approaches: To assess the self-optimization capability of our framework, we compare it against two competing approaches, i) *Round-robin*, in which the workload tasks are divided equally among all the available service providers and ii) a Pull-based First-Come-First-Served (P-FCFS) [50], in which service providers pull from a bag of tasks at the arbitrator whenever they become idle, work on them, and report the result. Round-robin is chosen for comparison to show the gains (in terms of application response time and battery drain) that can be achieved by exploiting the heterogeneity in computing capabilities of service providers. P-FCFS inherently exploits the heterogeneity in computing capabilities. However, it results in progressively faster devices completing a correspondingly higher number of tasks over time. It is also robust to service provider failures or loss in connectivity as it is purely pull based. However, due to lack of self-optimization, there is usually unfair battery drain at the service providers.

Experiment setup: We performed an experiment with three different service providers – a Samsung Galaxy Tab, a Motorola Atrix 2, and a HTC Desire HD – with significantly different computational capabilities and battery capacities (as shown in Table 3.1). The workload tasks are divided among these service providers based on the result of our resource allocation engine (with a deadline of 100 s) as well as on the two aforementioned competing scheduling mechanisms. The results in Fig. 3.6(a) (workload completion times) were obtained from one run while the results in Fig. 3.6(b) (residual battery capacity) were obtained from 100 consecutive runs of the same workload on the service providers (to achieve a significant battery drain). While the division of tasks among the service providers remains the same for all 100 runs as far as Round-robin and P-FCFS are concerned, the number of tasks to be worked on by the different service providers when our framework is used is determined by the resource allocation engine in every run.

Observations: Figure 3.6(a) shows the performance of the three approaches in terms of workload completion time. It can be observed that Round-robin misses the deadline and is



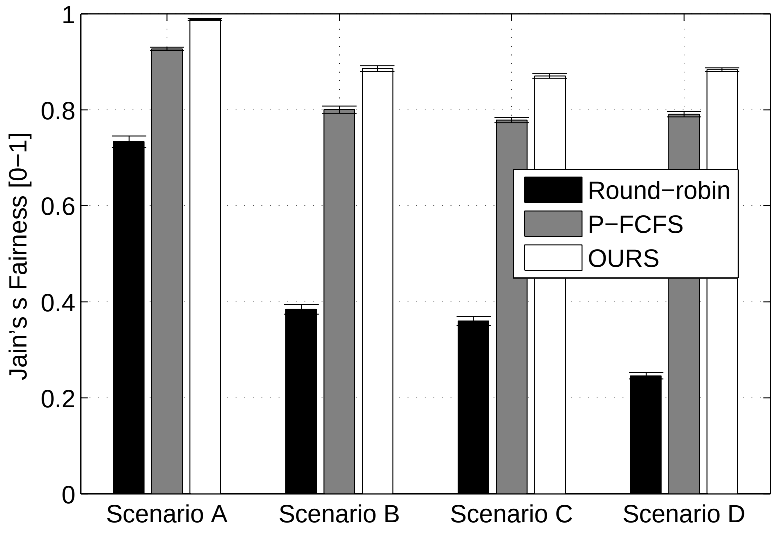
(a)



(b)

Figure 3.6: (a) Performance of proposed framework (in terms of task completion times [s]) versus P-FCFS and Round-robin approach; (b) Performance of proposed framework (in terms of battery drain [%]) versus P-FCFS and Round-robin approach.

the slowest of the three as it does not identify and exploit the heterogeneity of the available service providers in terms of their computational capabilities. P-FCFS is the fastest as more tasks are completed by faster devices. Our framework meets the specified deadline by exploiting the heterogeneity of the service providers. The main difference in performance



Combinations of service providers used in different scenarios

	Scenario A	**Scenario B**	**Scenario C**	**Scenario D**
# Tasks	75	150	225	300
Deadline [s]	60	60	60	60
# Galaxy Tab	1	2	3	5
# Atrix 2	1	2	3	3
# Galaxy S	1	1	1	4
# Optimus V	0	1	3	2
# HTC Desire	1	2	3	3
# Netbook	0	1	1	1
# Laptop	1	1	1	2

Figure 3.7: Performance of proposed framework (in terms of fairness) versus round-robin and P-FCFS approaches.

between our solution and P-FCFS can be observed in Fig. 3.6(b), which shows the residual battery capacity after 100 consecutive runs. P-FCFS does not appreciate the heterogeneity of devices in terms of battery capacity resulting in asymmetric battery drain.

In order to show the scalability of the proposed resource-allocation engine and its performance under difference operational scenarios (in terms of number and combination of service providers) as shown in the table in Fig. 3.7, we performed a simulation to ascertain the fairness in battery drain when each of the three task-scheduling mechanisms are employed. We used *Jain's fairness index* (1 being the highest and 0 being the lowest) as the measure of fairness.

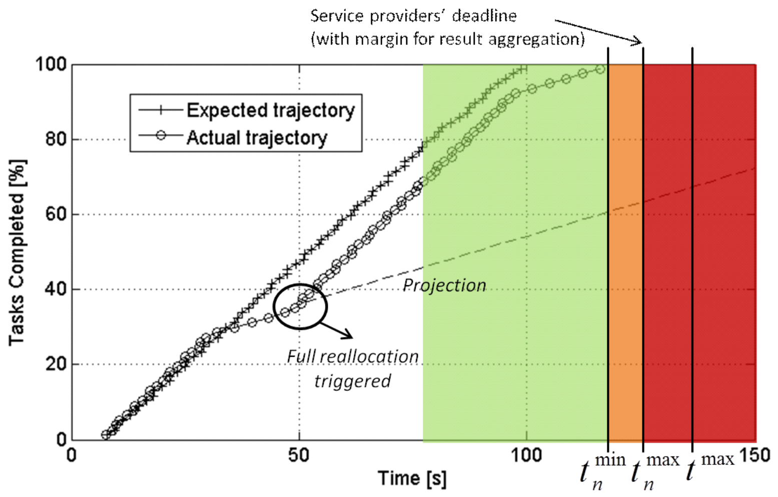
Simulation setup: The four scenarios in the table in Fig. 3.7 represent a progressive increase in the scale and the heterogeneity of the underlying service provider pool as well as the problem size (in terms of number of tasks). The scaling up is achieved by increasing the resolution of the object's image, which is the input to the object-recognition application. In order to determine the amount of battery drain while using the three task-scheduling mechanisms, we simulated 100 consecutive runs (for significant battery drain) of the workload. This procedure is referred to as one trial. We in turn performed multiple trials, each with a different starting condition in terms of available battery capacities in the service providers.

Observations: Figure 3.7 shows the average fairness in terms of residual battery capacity at the service providers after each trial. In order to obtain the confidence intervals, we performed multiple trials until we achieved a very small relative confidence interval (less than 10%). Our proposed solution achieves the best performance in terms of fairness in the residual battery capacity as it fully exploits the heterogeneity of the devices in the resource pool to achieve its objective while meeting the user-specified deadline.

3.4.4 Self-healing

Experiment setup: We performed an experiment with four service providers – two Samsung Galaxy Tabs, a Motorola Atrix 2, and a HTC Desire HD – to demonstrate the self-healing capability of our resource provisioning framework. The workload tasks were divided among the service providers based on the result of our resource allocation engine (with a deadline of 120s). One of the Samsung Tabs was disassociated from the arbitrator at the time instant 30s to show how the arbitrator uses the application waypoints as well as the service advertisements to identify anomalies (such as node failure, disassociation, etc.) and reacts to it by reallocating incomplete tasks to the available service providers.

Observations: Initially, the workload tasks are divided among three (one Samsung Galaxy Tab, one Motorola Atrix 2, and one HTC Desire HD) of the four service providers based on the result of our resource allocation engine. The second Galaxy Tab is not chosen initially due to its low residual battery capacity compared to the other devices. Figure 3.8 shows the trend of estimated task completion times as seen at the arbitrator over time. At the beginning of the workload execution, the task completion times follow the estimated task completion time (calculated using function g in our optimization problem). However, when one of the service provider (a Samsung Galaxy Tab) fails the trajectory of the task completion time violates the acceptable uncertainty region. This violation is detected by the



Details of the service providers used in the experiment.

Device type	# of devices	Residual battery %
Galaxy Tab	2	95,70
Atrix 2	1	80
HTC Desire	1	90

Figure 3.8: Demonstration of the use of application waypoints to handle uncertainty (detect node failure) and recover through re-allocation of resources.

arbitrator in the detection zone (the acceptable uncertainty region) and it reallocates the incomplete tasks among the available three service providers so to ensure that the workload is completed within the original estimated time (as seen in the recovery zone). During this reallocation, the second Galaxy Tab is used despite its low residual battery capacity as the other two devices alone cannot complete all the tasks within the specified deadline.

Chapter 4

Uncertainty-aware Mobile Application Workflow Management

4.1 Overview

Data and computing-resource uncertainty, if unchecked, may propagate up the data-processing chain and have an adverse effect on the relevance of the generated result (level of accuracy and timeliness). Prior work in the field of data-uncertainty management [33–35] and on computing-resource-uncertainty management [8, 27, 36] identifies and captures the limitations arising out of either imperfect/partial data or insufficient computing resources to provide not-so-accurate yet meaningful results within specified deadlines. However, these works focus on data- or resource-uncertainty management *in isolation* to prevent those from adversely affecting the result. However, uncertainty-aware computing in the mobile sensing and computing domain requires a *unified approach* (for both data as well as resource management) as it is affected by uncertainties arising out of both the sources.

In this chapter, we present a unified uncertainty-aware framework for management of data and computing resources in order to enable ubiquitous mobile-application workflows execution on mobile sensing and computing platforms and, hence, to generate actionable knowledge from raw data within realistic time bounds. Mobile-application workflows are typically made up of several data collection, computation, and result generation tasks with a pre-determined parallelism and order of execution. The relevance of the results from mobile-application workflows rely heavily on the quality and quantity of raw data coming from the underlying multi-modal sensing infrastructure as well as the computing resources available to execute them in real time. Note that a large amount of high-quality data does not guarantee good results as it increases the computational complexity and, hence, the computing-resource requirements. A small degree of uncertainty with computing resources (or data) can outweigh the benefits brought by data (or computing resources) necessitating a unified management approach.

We build our novel uncertainty-aware management framework on top of our autonomic mobile grid management middleware. The middleware aids in the organization of the heterogeneous sensing, computing, and communication capabilities of static and mobile devices in order to form an heterogeneous computing grid. When a user application places a service request to an arbitrator through a requester, it will specify i) the workflow that conveys the data-processing chain and ii) the acceptable level of approximation in the result (or an equivalent confidence measure for the response). We introduce a generalized yet powerful *workflow representation* to convey the different data sources and the data-processing tasks as well as to capture their relationships at the different stages of the workflow.

The arbitrator should determine *on the fly*, 1) the quantity of data to request from the DPs (*referred* to as the *task size*) along with 2) the most appropriate computational model (from a suite of models) to process the raw data, extract features, and generate a result (actionable knowledge) that satisfies the requester-specified constraint in terms of accuracy in the final output, and 3) the amount of computing resources to utilize from those made available by the RPs so that the result is delivered as soon as possible while ensuring fairness in battery drain at the RPs. It is evident that this is a complex combinatorial optimization problem as it involves the choice of the best combination of data, resources, and the model to generate results that satisfy requester-specified constraints. We propose a two-phase solution to make the aforementioned complex combinatorial problem tractable. In the first phase, based on the workflow, user-specified requirement as well as the knowledge of propagation of uncertainty, we decide on the computational models (tasks) to use and the corresponding task sizes. In the second phase, we assign the tasks at each stage to RPs. Our contributions in this chapter are as follows,

- We propose a simple yet powerful generalized workflow representation scheme to construct data-processing chains (tasks and dependencies) for ubiquitous mobile applications, which are composed multiple parallelizable and sequential tasks.
- We determine the appropriate task sizes at the different stages of the workflow by

estimating the propagation of uncertainty from raw data to the final result using *interval arithmetic* [51].

- We formulate the problem of assigning tasks to RPs at every stage of the workflow as a *combinatorial optimization problem with multiple bottleneck objectives* (application makespan and fairness in battery drain), and propose a polynomial-time stage-wise threshold-based heuristic called `Fast M-CBP` algorithm for the same.

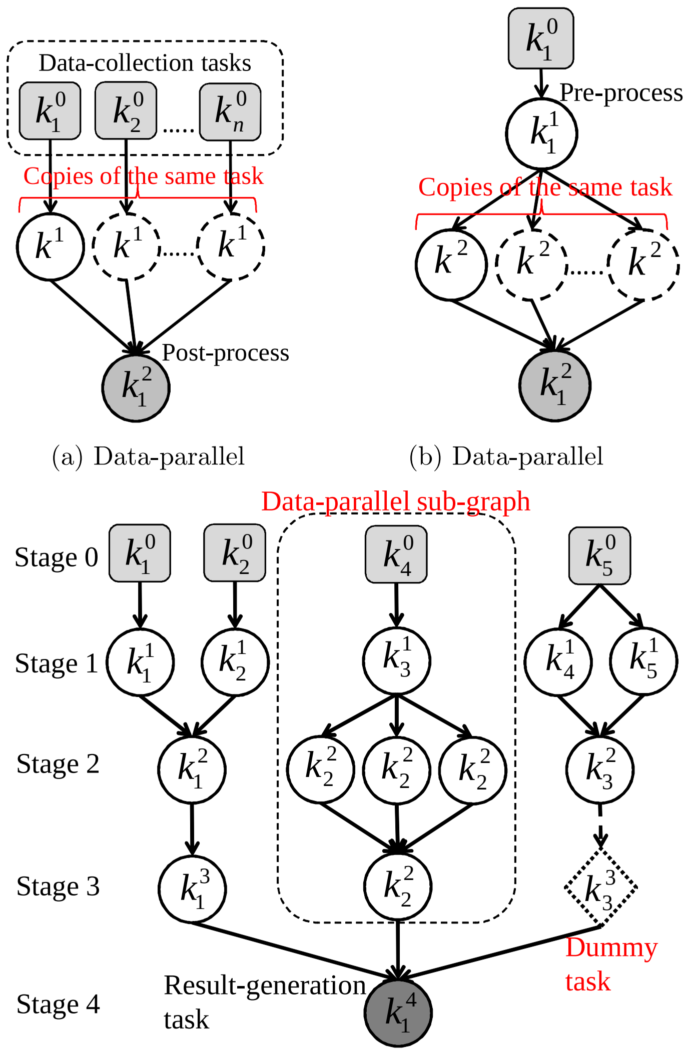
4.2 Workflows

Ubiquitous mobile applications are composed of multiple tasks whose order of execution is specified by a workflow. We have developed a simple yet powerful generalized workflow representation scheme to construct data-processing chains for mobile applications. First, we present our workflow representation scheme and then discuss biomedical workflows for ubiquitous health monitoring in detail.

4.2.1 Generalized Workflow Representation

Our generalized workflow is a *Directed Acyclic Graph* (DAG) composed of tasks (vertices) and dependencies (directed edges) as shown in Fig. 4.1. Tasks belong to one of the following three categories: i) data-collection task, ii) computation task, or iii) final result-generation task. These tasks are elementary and cannot be split further into micro-tasks. However, tasks can be grouped together to form macro-tasks. Each task is either in itself or a building block of a computational model that aids in the extraction of information from data.

The workflow is composed of multiple *stages* with a set of tasks $\mathcal{K}^i = \{k^i_j\}, j = 1, \ldots, |\mathcal{K}^i|$, to be performed at each stage i. Let M be the total number of stages, i.e., $i = 0, \ldots, M-1$. Stage 0 is composed entirely of data-collection tasks, which can be performed only at the DPs (sensing devices). The computation and final tasks (at any stage $i \geq 1$), however, can be performed at any RP (computing devices). Examples of computation and final tasks include frequency domain analysis, estimation of power spectral density (PSD), histogram analysis, linear combination, estimation of first order statistics, rule-based (threshold-based) decision making, etc.



(a) Data-parallel (b) Data-parallel

(c) Task-parallel

Figure 4.1: Workflows for data-parallel applications with n parallel tasks (a) when there are n separate data sources and (b) when data from a single source has to be divided into multiple chunks (pre-processing); (b) Workflow for a task-parallel application (with a data-parallel sub-graph in it).

This representation is powerful as it captures both *data-* and *task-parallel* applications. Data-parallel applications are also referred to as "embarrassingly parallel" applications in which an independent set of homogeneous tasks – working on disjoint sets of data – can be performed in parallel (preceded and succeeded by pre- and post-processing tasks, respectively) as shown in Fig. 4.1(a) and 4.1(b). Task-parallel applications, on the other hand, have a set of sequential as well as parallel tasks with pre-determined dependencies and degree of parallelism. A task parallel workflow may also have a data-parallel block built into it as shown in Fig. 4.1(c).

Different computational models may be suitable to accomplish a specific goal at a particular stage of the workflow depending on which of the following two criteria is more important: i) the time taken to achieve the goal or ii) how the model (used to achieve the goal) propagates uncertainty from the inputs to the output. For instance, if the goal is to generate one scalar output from ten scalar inputs, the specific model may be estimation of the maximum, minimum, median, mode, or mean (simple or weighted) depending on the time taken and/or on how each of these models propagate the uncertainty from the inputs to the output. Here, we assume that the computational models (i.e., the task types) in the workflow are fixed and known in advance.

Let $\mathcal{S}$ be the set of SPs, which includes both the data providers and the resource providers. The SPs that perform the tasks at stage $i-1$, where $i \geq 1$), serve as data sources $d^i_m, m = 1, \ldots, |\mathcal{K}^{i-1}|$ for the tasks that have to be performed at stage i. Tasks at stage 0 are performed at the DPs themselves (as mentioned earlier) and they do not have data sources as the data is generated locally. The data sources for tasks at stage 1 are the DPs themselves (SPs where stage 0 tasks are performed). The mapping of data sources (SPs at stage $i-1$ to tasks at stage $i \geq 1$ of the workflow is represented by the matrix $\mathbf{Q}^i = \{q^i_{sk}\}$, where $s \in \mathcal{S}$ and $k \in \mathcal{K}^i$. Note that $\mathbf{Q}^i$ captures only the mapping of data sources to tasks at stage i. The actual allocation of tasks at any stage $i \geq 1$ to SPs takes multiple objectives into consideration (as detailed in Sect. 4.3.2).

There are no dependencies between the tasks at a particular stage and, hence, they can be performed in parallel. Also, without any loss in generality, we assume that there can be dependencies only between tasks of consecutive stages in the workflow. Therefore, we use "dummy tasks", whose output equals the input without any propagation of uncertainty and whose cost of operation (in terms of time and battery drain) is zero, whenever we have dependencies between tasks of non-consecutive stages. In Fig. 4.1(c), task k^3_3 in stage 3 is a dummy task that has been introduced to break up the dependency between tasks k^2_3 and k^4_1 in non-consecutive stages 2 and 4, respectively. Our *structured* workflow representation is rich in information. In addition to the regular information like task types and data dependencies, it includes the following information: task identifiers, task sizes, quality and

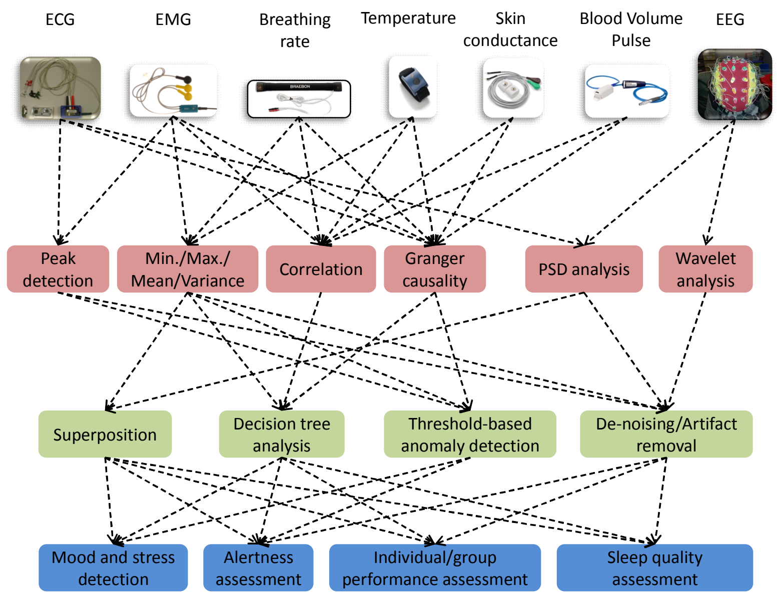


Figure 4.2: Sensor-based biomedical applications that constitute the larger class of ubiquitous healthcare applications.

quantity of inputs, the preferred interfaces to child and parent tasks, the implementation (when multiple exist), and the task criticality (either boolean or multiple degrees).

4.2.2 Biomedical Workflows

In order to understand the concept of mobile-application workflows and their constituent computational tasks, let us take the example of sensor-based biomedical applications (shown in Fig. 4.2). These applications are part of the larger class of ubiquitous mobile healthcare applications. Figure 4.2 shows the non-invasive sources (sensors) of different vital signs – Electrocardiogram (ECG), Electromyogram (EMG), breathing rate, temperature, skin conductance or Galvanic Skin Response (GSR), blood volume pulse, oxygen level, and Electroencephalogram (EEG) – along with the computational models for data analysis, data manipulation, and decision making. Some interesting ubiquitous mobile biomedical applications include mood and stress detection or stress-level rating as well as assessment of alertness, cognitive performance (individual as well as group), sleep quality, hypoxia (lack of oxygen) detection etc.

Data analysis here refers to the extraction of features such as first-order statistics, PSD, degree of correlation, and causality. Data-manipulation models may refer to data-cleaning

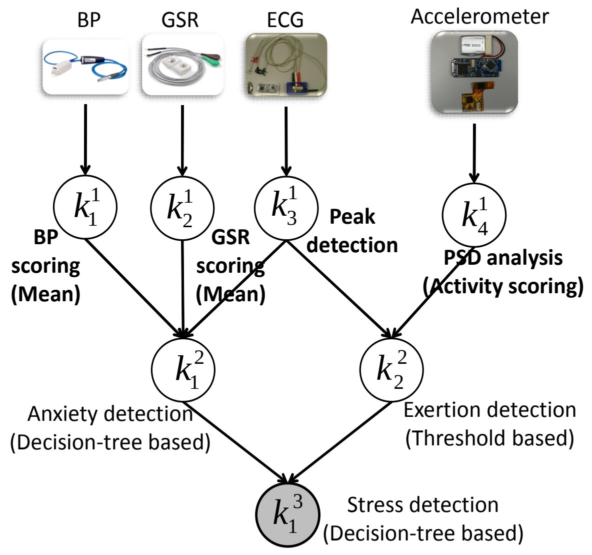

BP
GSR
ECG
Accelerometer
k_1^1
k_2^1
k_3^1
k_4^1
BP scoring (Mean)
GSR scoring (Mean)
Peak detection
PSD analysis (Activity scoring)
k_1^2
k_2^2
Anxiety detection (Decision-tree based)
Exertion detection (Threshold based)
k_1^3
Stress detection (Decision-tree based)

Figure 4.3: Typical workflows for sensor-based non-invasive stress-detection application. Dummy tasks are not shown in the figure for legibility.

tasks such as artifact removal or tasks for extraction of derived data (like weighted combinations) from multiple independent sources. Models for decision making include threshold-based anomaly detection and decision trees. Applications employ these tasks at different stages of their workflows (the data-processing chain) so to extract actionable knowledge from the raw data acquired in the field.

Stress detection and stress-level rating: Figure 4.3 depicts the workflow for stress detection and stress-level rating [52,53], which uses vital-sign data acquired from biomedical as well as kinematic sensors. Stage 1 of this workflow is purely composed of data-analysis tasks, whereas Stages 2 and 3 are composed of data-manipulation or decision-making tasks. In Stage 1, Heart Rate (HR), Blood Pressure (BP), GSR, and activity are each given a score $[0, 1]$ (0 corresponding to low, 1 to high) based on simple manipulations of the corresponding sensor outputs. Assigning these normalized scores requires domain knowledge and is in some cases subject specific (as in GSR and HR). Anxiety, physical exertion, and stress detection involve the use of decision-tree- or threshold-based models. Conversely, the model for anxiety-level scoring involves a weighted combination of HR, BP, and GSR scores. Similarly, physical exertion is scored by multiplying the activity and HR scores. Finally, stress level is determined using a weighted combination of scores for anxiety and physical exertion.

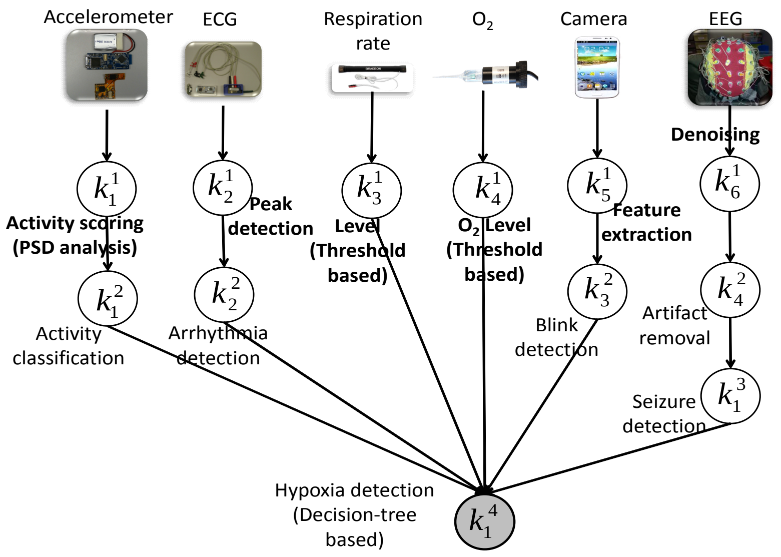


Figure 4.4: Typical workflows for sensor-based non-invasive hypoxia-detection application. Dummy tasks are not shown in the figure for legibility.

Hypoxia warning system: The hypoxia detection workflow shown in Fig. 4.4 takes as inputs raw data from diverse sensors (environmental and physiological). Stage-0 in Fig. 4.4 shows the suite of sensors used to determine onset of hypoxia in an individual. As onset of hypoxia symptoms at high altitudes is due to the reduction in oxygen partial pressure it is measured using environmental sensors like the oxygen partial pressure sensors (referred to as O_2 microsensor) while physiological sensors like pulse oximeters are used to measure the oxygen saturation in the individual's arteries. Data (like respiration rate, ECG, and EEG) from other non-invasive physiological sensors are also used to identify onset of hypoxic symptoms. ECG is used to detect any arrhythmia, whereas EEG is used to identify any seizures in the brain and blink rate. Separate eye-tracking sensor, like a high-resolution camera, may also be used to gather data about gaze direction or eye movements with very high accuracy.

In Stage 1 of the workflow in Fig. 4.4, data from both respiration rate and O_2 sensors goes through a threshold-based detection, ECG data goes through peak detection, and EEG data goes through a denoising stage. In Stage 2, the denoised EEG data is cleared of artifacts, while the heart rate deduced from the peaks in ECG data is used for arrhythmia detection. In the same stage, the human activity and eye blinks are detected using data from accelerometers and the high-resolution camera. In Stage 3, the processed EEG data is scanned for symptoms of seizures and finally in Stage 4 all this information is fed to a decision-tree model that determines the severity of hypoxia in the soldiers' body and helps issue a warning for appropriate medical action.

4.3 Methodology

When the requester places a service request to an arbitrator (initiated by an automated application or the user of an application), it will specify i) the workflow that conveys the data-processing chain using the generalized workflow model and ii) the acceptable level of approximation in the result (or an equivalent confidence measure for the response). Let the user-specified acceptable level of uncertainty in the result be γ. The arbitrator should solve a *complex combinatorial problem* to determine the best combination of computational models (for tasks), quality and quantity of data (task sizes), and the computing resources to use such that the quality of the generated result is maximized. Quality of the result is determined by two factors, accuracy and timeliness. While the choice of tasks and task sizes have a bearing on the accuracy as well as the timeliness of the result, the choice of computing resources has an effect only on the timeliness. We exploit this fact and propose a *two-phase solution* to make the aforementioned combinatorial problem tractable.

In Phase I, the arbitrator first derives the maximum acceptable uncertainty at the output of every task at every stage of the workflow (i.e., the γ_j^is, where $i = 1, \dots, M-1$ and $j = 1, \dots, |\mathcal{K}^i|$) based on the user-specified γ and the knowledge of the workflow. Then the arbitrator determines sampling duration (i.e., the task sizes) so that the uncertainty bound on the output of every task at every stage is not violated. In Phase II, the allocation of tasks at every stage of the workflow to SPs is determined with the following two objectives in mind: minimization of total response time "and" minimization of maximum battery drain among all SPs in the heterogeneous computing grid. Even though the task allocation is determined in one shot, execution of the workflow is carried out in a *stage-wise* manner. The stage-wise approach serves as a *natural checkpointing mechanism*, which helps avoid propagation of uncertainty in the event of SP failures.

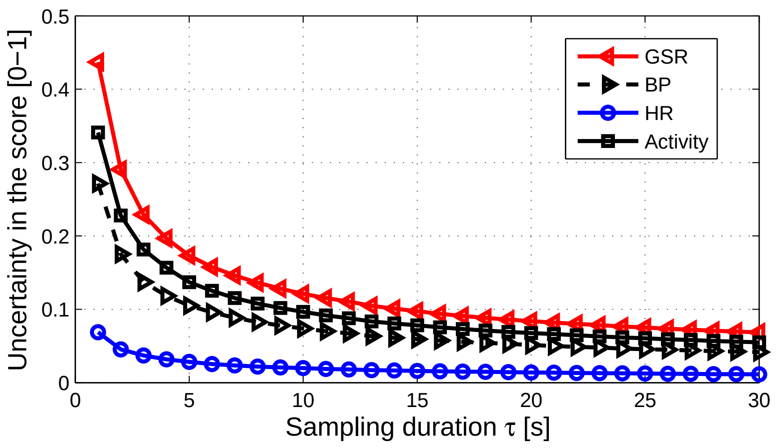


Figure 4.5: The decrease in uncertainty γ (in terms of width of the interval within which the score lies) with increase in sampling duration τ. Heart Rate (HR), Blood Pressure (BP), Galvanic Skin Response (GSR), and Activity are each given a score [0,1] (0 -low, 1-high) based on the sensor output. Variability in such output necessitates representation of the score in terms of an interval. The higher the variability in sensor output, the larger the uncertainty in the score.

4.3.1 Phase I: Propagation of Uncertainty

The end-user application conveys the acceptable level of uncertainty in the final result in terms of the "maximum width γ of the interval" within which the actual result lies. An interval (i.e., a range of possible values) is a compact way of representing the uncertainty associated with a particular quantity. The quantities of interest can be direct sensor outputs (e.g., temperature, skin conductance) or derived quantities from the raw sensor data (e.g., heart rate, blood pressure). When intervals are used to represent quantities, the mathematical computations on those quantities also need to be performed on intervals instead of fixed values. The branch of mathematics that deals with intervals instead of fixed values is known as *interval arithmetic* [51].

Confidence Intervals (CIs): We leverage interval arithmetic to study the effect of the "variability in sensor data" on the "variability in the final result". Figure 4.5 shows the decrease in uncertainty γ – in terms of width of the interval within which the HR, BP, GSR, and Activity scores lie – with increase in sampling duration. As detailed in the previous section, HR, BP, GSR, and Activity are each given a score (between 0 and 1) but in the form of an interval (e.g., $[0.45, 0.55]$). The higher the variability in sensor output, the larger the uncertainty in the score. It is important to note that the minimum and maximum

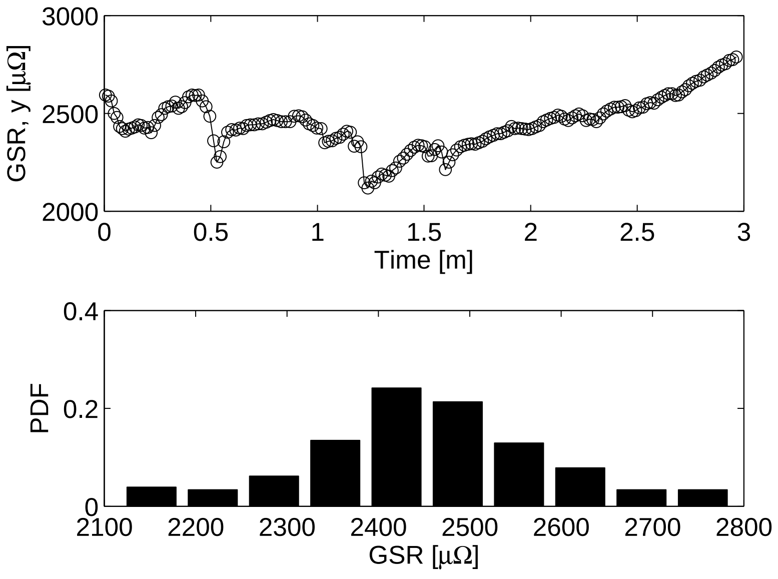
Table 4.1: Interval-arithmetic rules for basic mathematical operations and monotonic functions in one variable

Operation	Interval of the result
$[x_1, x_2]\diamondsuit[y_1, y_2]$, where $\diamondsuit \in \{+, -, *, /\}$	$[\min\{x_1\diamondsuit y_1, x_1\diamondsuit y_2, x_2\diamondsuit y_1, x_2\diamondsuit y_2\},$ $\max\{x_1\diamondsuit y_1, x_1\diamondsuit y_2, x_2\diamondsuit y_1, x_2\diamondsuit y_2\}]$
$[x_1, x_2]^n$	$[x_1^n, x_2^n]$, if $x_1 \geq 0$ and odd $n \in \mathbb{N}$ or even $n \in \mathbb{N}$, or if $x_1 \geq 1$ and $0 \leq n \leq 1$ $[x_2^n, x_1^n]$, if even $n \in \mathbb{N}$, $x_2 < 0$ $[0, \max\{x_1^n, x_2^n\}]$, if even $n \in \mathbb{N}$, otherwise
$\log_b[x_1, x_2]$	$[\log_b x_1, \log_b x_2]$, $\forall\ x_1, x_2 > 0$, $b > 1$

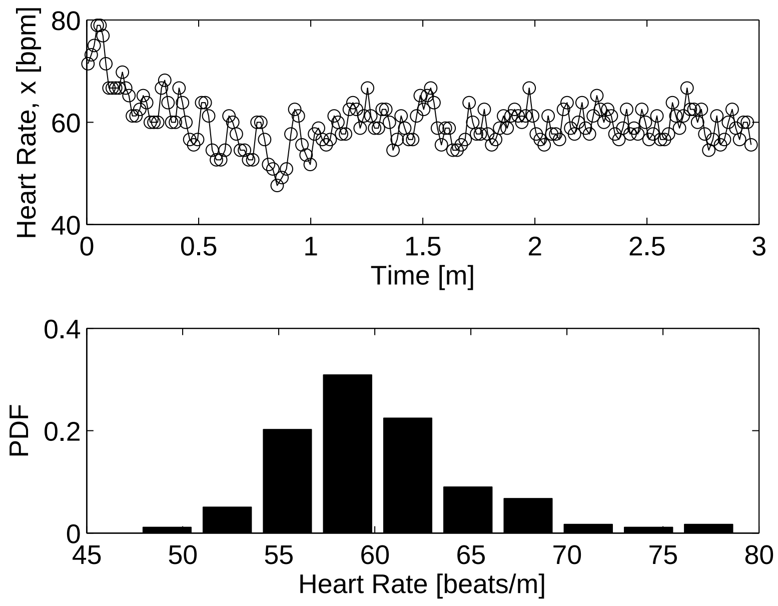
values of the raw sensor data, which were used for computing the score, correspond to the boundaries of the 95% confidence interval of samples instead of the absolute maximum and minimum values as in the case of traditional interval arithmetic. The actual width of the 95% CI is large when the sampling duration is small and viceversa. Usage of boundaries of 95% CIs instead of that of the entire range of values makes our method resilient to outliers, which unnecessarily widen the intervals (especially in the case of small sample sizes).

The use of 95% as "prediction interval" (or "reference interval") is a standard practice in the interpretation of biochemical data [54]. However, as we are interested in the accuracy of the estimate of interest (in our case, mean), we use 95% confidence interval around the mean and not the reference range (which may be large). We assume that estimates of the width of the 95% confidence intervals for different sample sizes (corresponding to different τ's) are available. This is a realistic assumption as every sensor can be calibrated offline, which needs to be done only once. These estimates can be maintained locally at the DPs and communicated to the arbitrator when requested, or can be maintained at the arbitrator. In our case, we assume that the arbitrator is in possession of the interval estimates. Also, the assumption that biomedical sensor measurements and derived quantities are *normally distributed* (or log-normal) is justified based on our observation of the distribution of exogenous biological markers like GSR (direct sensor data) and heart rate (derived from real ECG data), as shown in Figs. 4.6(a) and 4.6(b), respectively. It has been observed that most endogenous biological markers including blood glucose and ionized calcium also follow normal or log-normal distributions [54].

Uncertainty propagation using interval arithmetic: Workflow tasks are represented mathematically using a combination of the basic operations listed in Table 4.1, where



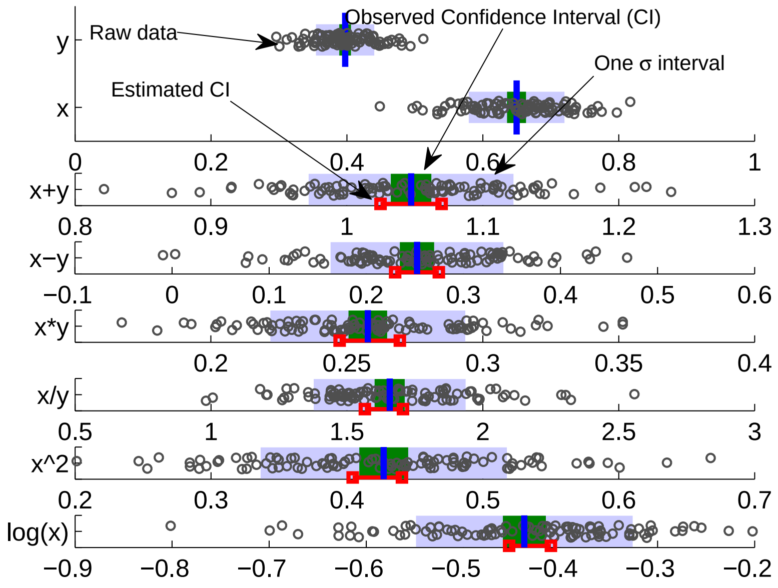
(a)



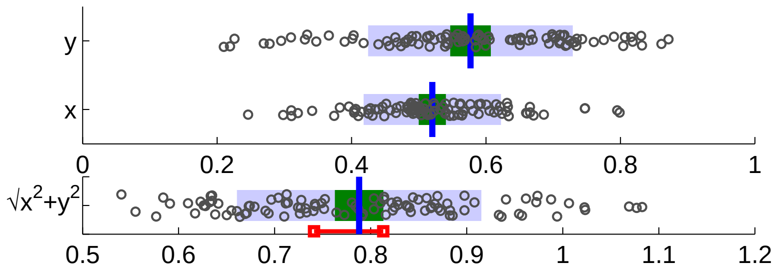
(b)

Figure 4.6: Empirical evidence validating the assumption about the Probability Distribution Function (PDF) of (a) direct sensor measurements (GSR) and (b) a derived quantity (heart rate derived from ECG data). Both sets of values are normally distributed around the mean.

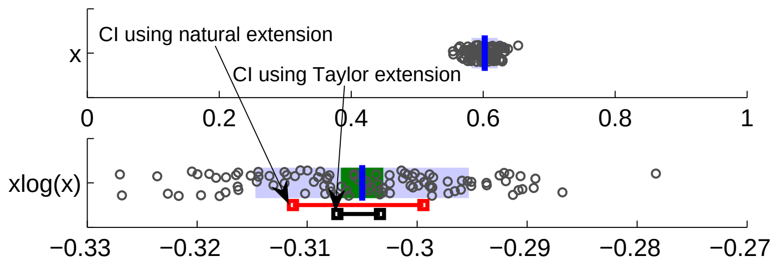
$\diamondsuit$ represents the basic mathematical operations $+, -, *, /$. Once the confidence interval estimates at the different DPs are known (for a particular τ), the rules of interval arithmetic are leveraged to propagate the uncertainty up the workflow to the final result. Figure 4.7



(a) Basic operations and simple functions



(b) Euclidean distance using natural extension



(c) $x \log(x)$ using natural *and* Taylor extensions

Figure 4.7: (a) Operations listed in Table 4.1 are performed on several realizations of two normally distributed random variables, x and y. The theoretical 95% CI obtained using the rules (shown using red lines) are consistently more conservative than (and close to) the measured 95% CI (depicted as green zone). The blue zone is the one-sigma interval, while the gray circles are the raw values; (b) Illustration of natural interval extension to find the output interval for Euclidean norm on two variables; (c) Illustration of improvement in precision when using Taylor interval extension over natural interval extension for computing logarithmic utility.

shows empirical evidence validating the use of 95% confidence intervals for interval arithmetic. Basic mathematical operations (addition, subtraction, multiplication, division, and Euclidean distance from the origin) were performed on several realizations of two normally-distributed random variables, x and y. The theoretical confidence intervals obtained using the rules of interval arithmetic in Table 4.1 are consistently more conservative than (and close to) the measured 95% confidence intervals of the results, as shown in Fig. 4.7(a). The results do not differ in the case when the two intervals overlap.

Special conditions: Note that there are, however, specific conditions under which certain rules of interval arithmetic hold when using CIs. The division rule applies when the coefficient of variation $c.v = \frac{\sigma}{\mu}$, i.e., the ratio of standard deviation σ to mean μ, of the denominator is less than 0.09 and that of the numerator is greater than 0.19, and when the correlation between the two random variables is less than 0.5 [55]. The normal approximation of the result and, hence, the use of CI, is valid under these conditions. When these conditions are not met, the Geary-Hinkley transformation [55] provides normal approximation for the result as long as the $c.v$ of the denominator is less than 0.39 and that of the numerator is greater than 0.005. Similarly, the $c.v$ requirements of the random variable is stringent for the exponent rule, $a^{[x_1,x_2]} = [a^{x_1}, a^{x_2}], a > 1$, thus necessitating a careful application of interval arithmetic with CIs.

Extension to general functions: Even though Table 4.1 lists only the elementary mathematical operations and monotonic functions, interval methods can be extended to general functions in two possible ways, *natural extension* and *Taylor extension*. In natural extension, a function is represented using the elementary mathematical operations and monotonic functions in one variable. Figure 4.7(b) depicts an example of using natural interval extension to successfully compute the output interval of Euclidean distance, i.e., $\sqrt{x^2 + y^2}$, which is a function commonly used in distance- or threshold-based models for anomaly detection (e.g., in stress and hypoxia detection in the case of biomedical applications). Natural extension works fine in most cases except when there are "dependency issues". Dependency issues arise when a variable occurs more than once in the expression, e.g., as in $f_1(x) = x^2 - x$ and $f_2(x) = x\log(x)$, and when the two occurrences are treated as if they were independent variables. Quadratic, like $f_1(x)$, as well as logarithmic utility, like

$f_2(x)$, functions are common in biomedical workflows as they are used to model the rate of growth or death of a population (e.g., cancer cells, pathogens). For such functions, the bounds provided by natural extension is relaxed and we have to transform the expression to have the variable appear only once, e.g., $f_1(x) = x^2 - x = (x - 1/2)^2 - 1/4$, and get a tighter bound. This transformation is, however, not possible for many functions as in the case of $f_2(x) = x \log(x)$, for which, therefore, we need to use Taylor interval extension in order to obtain a tighter bound.

Let $[x]$ represent an interval $[x_1, x_2] \in [\mathbb{R}]$ and use $[\mathbf{x}]$ to represent a vector of intervals $([x]_1, \ldots, [x]_n) \in [\mathbb{R}]^n$. The Taylor interval extension of a $m+1$ times differentiable function f, $[f] : [\mathbb{R}]^n \to [\mathbb{R}]$ over $[\mathbf{x}]$ is defined as,

$$[f][\mathbf{x}] := f(\mathbf{y}) + \sum_{i=1}^{m} \frac{1}{i!} f^{(i)}(\mathbf{y}) \left([\mathbf{x}] - \mathbf{y}\right)^i + [r]([\mathbf{x}], \mathbf{y}), \tag{4.1}$$

for some $\mathbf{y} \in [\mathbf{x}]$, where $f^{(m)}(\mathbf{y})$ is the m^{th}-order differential of f at the point $\mathbf{y}$ and $[r]$ is the interval extension of the Taylor reminder. Typically, $[\mathbf{y}]$ is chosen to be the midpoint and natural interval extension is leveraged to find the reminder. The special case of Taylor extension of degree $m = 0$ is called mean-value form. Note that when there are no dependency issues, both natural and Taylor extension yield very similar results. Figure 4.7(c) illustrates how tighter CI bounds can be obtained using Taylor interval extension rather than the natural interval extension for $f_2(x)$.

The two cases: We make the observation that in Phase I there are two possible cases, *C-fusion* and *R-fusion*, based on the *type of information fusion* performed by the application, and develop specific solutions for both cases. Information fusion from multiple sources can be classified as either *complementary, cooperative* (both of which we call "C-fusion") or *redundant* ("R-fusion"), based on the relationship among the different sources [56]. While C-fusion aims at completeness either through the creation of broader or new (derived) information, R-fusion aims only at increasing the reliability and accuracy. Also, in R-fusion, data from the redundant source can either be used as side information to improve the overall information quality or as a substitute of the primary source (intermittently) in order to increase the overall lifetime of the sensing infrastructure.

Algorithm 1 Find_Quantity (for Case C-fusion)

```
Input: Max. uncertainty $\gamma$ and workflow $\mathfrak{W}$
Output: Sampling duration $\tau$

  for $\tau = \tau_{min}$ to $\tau_{max}$ do
    Estimate the width of the 95% CI
    Propagate uncertainty up the workflow to estimate $\gamma'$
    if $\gamma' \leq \gamma$ then
      return $\tau$
    else
      Continue
    end if
  end for
```

The *C-fusion Case* covers the following scenarios: there is only one sensor for every data type, or there are multiple sensors for each data type and the data from all the different sources are fused together (via complementary or cooperative fusion) in the workflow. The complementary or cooperative fusion operations are treated as computational tasks in the application workflow. Figure 4.3 illustrates examples of complementary as well as cooperative information fusion. Specifically, the use of ECG, GSR, and BP to detect anxiety is an example of complementary fusion, whereas the use of anxiety and exertion information to detect stress is an example of cooperative fusion. The assumption here is that every sensor's data is useful for the computation of the final result.

In the C-fusion Case, the objective in Phase I is to determine only the sampling duration (and, hence, the quantity of data) at each sensor in order to minimize the uncertainty in sensor data or in the derived quantity of interest and, therefore, restrict the variability in the final result to a pre-specified γ. Sampling duration (with a priori knowledge of the sensor sampling rate) determines the number of samples used to calculate CI. We perform a linear search (as shown in Algorithm 1) for the most appropriate sampling duration (τ) by leveraging interval arithmetic to study how the uncertainty propagates up the workflow as described earlier. If this estimate in the result uncertainty is smaller than the pre-specified γ, then the sizes of the tasks at all the M stages of the workflow (i.e., $\mathbf{R^i} = \{r^i_j\}, j = 1, \ldots, |\mathcal{K}^i|$, for $i = 0, \ldots, M-1$) can be calculated using sampling duration τ, knowledge of the sampling rate of the different sensors, and sensor-output data type.

The *R-fusion Case* covers the following scenario: there are multiple sensors for each

Algorithm 2 Find_Quality (for Case R-fusion)

Input: γ, $\mathcal{D}$, and workflow $\mathfrak{W}$

Output: Best combination of DPs d^* and τ_{p^*}

```
$d^* = NONE$
for every $d \in \mathcal{D}$ do
  $\tau_d$ = Find_Quantity($\gamma$, $\mathfrak{W}$)
  if MaxBatteryDrain($d$) $\leq$ MaxBatteryDrain($d^*$) then
    $d^* \leftarrow d$
  end if
end for
return $d^*$, $\tau_{d^*}$
```

data type and data from one of the different sources is used (redundant fusion) in the workflow. Either ECG or wrist-located pulse detector can be used to determine the heart rate. Similarly, information about the physical activity can be obtained either using an accelerometer or a gyroscope. These are examples of redundant fusion. The assumption here is that information from one of the redundant sources is sufficient for the computation of the final result. The data sources can be managed in such a way as to increase the overall lifetime of the sensing system.

In the R-fusion Case, the objective in Phase I is to determine the most appropriate sensor (i.e., the data provider and, hence, the *quality of data* to collect) for each data type *as well as* the sampling duration at those sensors (*quantity of data*) so to restrict the variability in the final result to a pre-specified γ. Similar to the C-fusion Case, we perform a linear search (as shown in Algorithm 2) for the most appropriate sampling duration (τ) by leveraging interval arithmetic to study how the uncertainty propagates up the workflow for every combination ("without redundancy") of data sources (represented by set $\mathcal{D}$). Then the combination $d^* \in \mathcal{D}$ that minimizes the maximum battery drain (in terms of percentage) among the data sources is chosen. This way the algorithm determines *both* the quantity and quality of data. Redundant data sources usually differ in terms of their energy usage (rate of battery drain) and precision. A naive policy may always prefer the more precise sensor or the more energy-efficient sensor (the one with the least battery drain). However, such policies lead to an unfair usage of resources and may ultimately adversely affect the overall lifetime of the sensing system.

Anomalous situations: The decisions regarding data quality and quantity made in Phase I, while applicable in the *steady state*, may not be so under non-idealities. By non-idealities, we refer to anomalous situations in which i) the sensor is corrupted and reports highly fluctuating values, and ii) the phenomenon being sensed is in the *transient state*. In both these situations, the sensor measurements may not follow a normal distribution (they may in fact follow a *bimodal* or, in general, a *multimodal* distribution). Our solution handles such situations as follows. At the end of Phase I, the arbitrator provides the DPs not only the τ but also the target uncertainty level (i.e., the width of the CI of measurements). The DPs collect data until the uncertainty requirement is satisfied. If this requirement is not satisfied, they raise an alarm and notify the arbitrator. How then the arbitrator chooses to handle such anomalous situations (either through more sampling or choosing a different DP for that data type) is out of the scope of this dissertation.

4.3.2 Phase II: Stage-wise Multi-objective Optimization

Once the tasks and task sizes at all the stages of the workflow are determined in Phase I, they are allocated to SPs for execution in Phase II. Let $\mathcal{K}$ be the set of workflow tasks that have to be completed at a particular stage of the workflow. As we address the problem of workflow task allocation to SPs one stage at a time, we do not use an index to represent the current stage of the workflow. Service discovery at the arbitrators is achieved through service advertisements from the SPs. In addition to basic information about location, duration of availability (i.e., start t_s^{in} and end t_s^{out} times of the availability), and residual battery capacity (e_s^{adv} [Wh]) at each SP $s \in \mathcal{S}$, advertisements from DPs include the types of sensors and first-order statistics about the measurements while the ones from RPs include information about the amount of computing (γ_s^{cpu} [normalized CPU cycles]), memory (γ_s^{mem} [Bytes]), and communication (γ_s^{com} [bps]) resources available for use.

Let the time taken to complete a *unit task size* of task $k \in \mathcal{K}$ at SP s be t_{sk}^{comp} [h] and let the time taken to transfer a unit block of data (in our case 0KB) from one SP s_1 to another s_2 be $t_{s_1 s_2}^{comm}$ [h]. These parameters together with the task sizes give the total amount of time required to execute the different tasks at the different SPs. Let the instantaneous power drawn by every workflow task k when running on a specific SP s be w_{sk}^{comp} [W] and let the power drawn by the network interface at the SPs be w_s^{com} [W]. We assume that the arbitrator is aware of these parameters as there is only a finite number of types of SPs and the types can be known in advance.

Multi-objective Combinatorial Bottleneck Problem (M-CBP): Based on all the aforementioned information the arbitrator determines the allocation matrix $\mathbf{A} = \{a_{sk}\}$ (the optimization variable (4.2)) that conveys whether a task k has been allocated to a SP s or not. In this optimization problem, there are two objectives (4.3): i) *minimization of the maximum battery drain at all the SPs* (in terms of percentage) and ii) *minimization of maximum time taken by SPs to complete all the tasks that have been assigned to them.* If the problem were to have only the first objective, it would imply minimization of the maximum battery drain. On the other hand, if the problem were to have only the second objective, it would imply minimization of the total response time. However, the two-objective problem strives to minimize response time while ensuring fairness in terms of battery drain at the SPs. This fairness maintains the heterogeneity of the resource pool for longer periods by maximizing the lifetime of every single SP. This is a combinatorial optimization problem with multiple bottleneck objectives and in short is referred to as a Multi-objective Combinatorial Bottleneck Problem (M-CBP) [57].

$$\textbf{Find}: \quad \mathbf{A} = \{a_{sk}\}, \; s \in \mathcal{S}, k \in \mathcal{K}; \tag{4.2}$$

$$\textbf{Min:} \quad \begin{cases} \max_{s\in\mathcal{S}} \sum_{k\in\mathcal{K}} c^1_{sk} \cdot a_{sk}, \\ \max_{s\in\mathcal{S}} \sum_{k\in\mathcal{K}} c^2_{sk} \cdot a_{sk} \end{cases} \tag{4.3}$$

$$where, \; c^1_{sk} = \frac{r_k}{e^{adv}_s} \cdot (w^{comp}_{sk} \cdot t^{comp}_{sk} + $$

$$+ w^{comm}_s \cdot \tilde{t}^{comm}_{sk}), \tag{4.4}$$

$$c^2_{sk} = r_k \cdot (t^{comp}_{sk} + \tilde{t}^{comm}_{sk}); \tag{4.5}$$

$$\textbf{S.t.:} \quad \sum_{s\in\mathcal{S}} a_{sk} = 1, \; \forall k \in \mathcal{K}. \tag{4.6}$$

Here, $\mathbf{C}^1 = \{c^1_{sk}\}$ and $\mathbf{C}^2 = \{c^2_{sk}\}$ are the cost matrices corresponding to the two objective functions and the sole constraint (4.6) ensures that a task is allocated to one and only one SP.

The cost matric $\mathbf{C}^1$ captures the battery drain (for communication as well as computation) incurred while executing a particular task at a particular SP. Similarly, cost matrix $\mathbf{C}^2$ captures the time spent (for communication as well as computation) when a particular task is performed at a particular SP. Also,

$$\tilde{t}_{sk}^{com} = \sum_{s' \in S} t_{ss'}^{com} \cdot q_{s'k}, \; \forall s \in \mathcal{S}, k \in \mathcal{K}. \tag{4.7}$$

Here, the product of the communication delay matrix and the Q matrix (which captures the dependencies between tasks across successive stages) allows us to extract the information about the communication delays when a certain task is assigned to a certain SP.

`Fast M-CBP`: This algorithm is a polynomial-time heuristic that we propose to solve the M-CBP. This heuristic follows a threshold-based approach to determine the best possible allocation that achieves *Pareto optimality* (where one objective cannot be improved further without adversely affecting the other). Let $\mathcal{E}$ be the set of all possible combinations of the different thresholds. In our case, every $\varepsilon = \{\epsilon_1, \epsilon_2\} \in \mathcal{E}$ is two dimensional as we have only two cost matrices, one for the battery drain ($\mathbf{C}^1$) and one for the time taken ($\mathbf{C}^2$). With the aid of the thresholds, the two cost matrices are collapsed into one cost matrix $\mathbf{C}^*$ with binary weights (as shown in Algorithm 3). Then a simple minmax objective, shown in Algorithm 4, is used to find a feasible allocation using $\mathbf{C}^*$. Feasibility is not only determined by whether every task has been successfully allocated to a SP or not. `Fast M-CBP` searches for a feasible solution, an allocation $\mathbf{A}$ such that each task is allocated to *one and only one* SP, and the maximum number of tasks at a particular SP does not exceed β.

The arbitrator derives β from the knowledge of the following statistics about the SPs (maintained at the arbitrator) at a particular locality: the average arrival rate $\widetilde{\lambda}$ of SPs, their average sojourn duration $\widetilde{T}$ (whose inverse is called *churn rate*), and the average number of SPs $\widetilde{N}$. The relationship between these three numbers is given by the Little's Theorem, $\widetilde{N} = \widetilde{\lambda} \cdot \widetilde{T}$. When $\widetilde{T}$ is comparable to the average task execution time, then β is set to 1 in order to minimize the number of incomplete tasks (when SPs leave). However, when $\widetilde{T}$ is larger than the average task execution time (e.g., a few multiples), a higher number of tasks can be packed at fewer SPs with minimal risk of task incompletions. As the algorithm

Algorithm 3 `Fast M-CBP`: a heuristic for M-CBP

```
Input: $\mathbf{C}^1$, $\mathbf{C}^2$, $\mathcal{E}$, $\beta$
Output: An optimal solution $\mathbf{A}^*$ on the Pareto front

for every $\varepsilon \in \mathcal{E}$ do
    {Construct the bi-adjacency matrix $\mathbf{C}^*$}
    for every $s \in \mathcal{S}$ and $k \in \mathcal{K}$ do
        if $c^1_{sk} \leq \epsilon_1$ and $c^2_{sk} \leq \epsilon_2$ then
            $c^*_{sk} \leftarrow 1$
        else
            $c^*_{sk} \leftarrow 0$
        end if
    end for
    $[Feasibility, \mathbf{A}] \leftarrow$ MINMAX_ALLOCATE($\beta$, $G[\mathbf{C}^{*T}]$)
    if $Feasibility ==$ False then
        Continue
    else
        $\mathbf{A}^* \leftarrow \mathbf{A}$
        Break
    end if
end for
return $\mathbf{A}^*$
```

Algorithm 4 MINMAX_ALLOCATE: fair task allocation

```
Input: $\beta$, $G[\mathbf{C}] = \{\mathcal{K}, \mathcal{S}; E\}$, where $\mathbf{C} = \{c_{ks}\}$, $k \in \mathcal{K}$, $s \in \mathcal{S}$
Output: $Feasibility$ and $\mathbf{A} = \{a_{sk}\}$

Find $\mathbf{A}$ s.t. the max. no. of tasks assigned to SP $s$ is minimized
$\beta^* = \max_{s \in S} \sum_{k \in K} a_{sk}$
if $\beta^* \geq \beta$ then
    return False
else
    return [True, $\mathbf{A}$]
end if
```

is threshold based, all the feasible solutions it provides will be Pareto optimal. Therefore, instead of finding all possible Pareto optimal solutions (the so-called *Pareto front*), we force an exit once the first feasible solution is found. This along with a careful choice of the granularity of the discrete thresholds can drastically improve runtime performance.

4.3.3 A Note on Cyclic Workflows

The example workflows discussed so far are all *Directed Acyclic Graphs* (DAGs). However, there are many applications (or algorithms) that are repetitive or iterative in nature and, hence, have cycles in their workflow representation. Note that our two-phase solution is

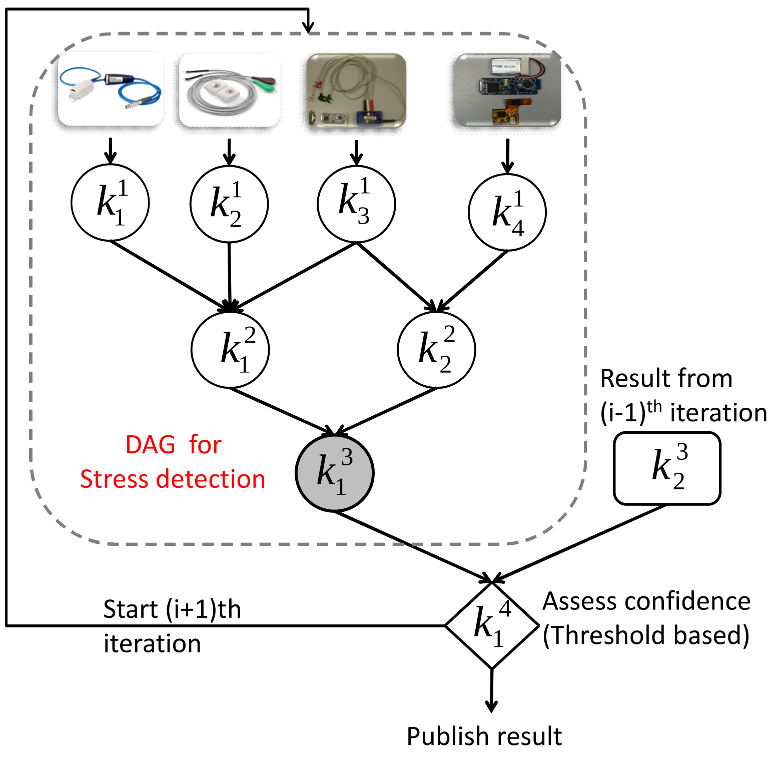


Figure 4.8: Example of a cyclic workflow that performs stress detection iteratively in order to improve confidence (for reliability) in the final published result.

also capable of handling real-time in-situ processing of *cyclic workflows*. As our solution is stage wise, we break the cycle and transform the iterative or repetitive workflow into a *sequence of DAGs*. The number of iterations in the algorithm determines how many times the DAG is executed. In some cases, the number of iterations is known a priori while in some other cases, after the first iteration, subsequent DAGs are executed only on demand. The aforementioned DAG-sequencing technique applies even in the case of workflows that have cycles only in a subgraph. In such cases, the workflow (graph) is first partitioned into component workflows (subgraphs), and then the DAG sequencing is performed only in the cyclic subgraph.

To understand this concept better, take the case of “reliable” stress detection and stress-level assessment of personnel under extreme environments (like a scene of disaster or the battlefield). The uncertainty associated with biomedical and kinematic data in such operating conditions will be high due to high mobility and intermittent connectivity. Therefore, the overall application workflow for reliable stress detection will be a *cyclic* one in which

Table 4.2: Average unit-task execution times and battery drain of computing devices in the testbed

Task	**Dell Inspiron**	**Dell Netbook**	**Samsung Galaxy Tab**	**Samsung Galaxy S3**	**Motorola Atrix 2**	**HTC Desire HD**	**LG Optimus**
Battery capacity (Wh/V)	55.5/11.1	50/11.1	28/4	5.7/3.8	6.6/3.8	5.3/3.8	5.7/3.7
Peak detection (s/mA)	1/23	2.5/42	3.6/36	3.1/40	6.5/45	6.8/48	10.1/40
Correlation (s/mA)	1.2/25	2.5/42	3.68/38	3.3/40	6.8/47	6.8/47	10.3/42
PSD (s/mA)	1.5/25	2.9/50	4.5/47	3.7/50	7.2/50	7.5/49	14.8/56
Granger causality (s/mA)	2.8/50	6.1/85	9.6/78	6.8/85	12.7/80	13.3/83	23.6/75
ICA (s/mA)	14.4/80	28.5/130	30.6/128	33.5/120	45.1/125	47.7/130	61.2/135

multiple rounds of stress detection are performed to increase the confidence in the result, as shown in Fig. 4.8. This cyclic workflow is transformed into a sequence of DAGs. Here, the number of DAGs in the sequence is not known a priori.

4.4 Evaluation

We have implemented a small-scale prototype of the proposed framework and performed an empirical evaluation. We have also used simulations to show the scalability of the proposed framework. In the following sections, firstly, we present details about our experiment methodology. Then, we discuss specific experiment scenarios and the results that demonstrate the benefits of data-uncertainty awareness (Phase I) and the performance of our proposed heuristic for the multi-objective optimization problem (Phase I and Phase II).

Devices and application profiling: Our testbed consists of Android- and Linux-based mobile devices with heterogeneous capabilities (summarized in Table 4.2). As the objectives of the optimization problem are concerned with fairness in battery drain and makespan, the amount of battery drain in the RPs as a result of running workload tasks as well as the task execution times need to be profiled in advance. However, the usage of actual Watt-hour (Wh) values will result in unfair usage of RPs with a higher battery capacity. Hence, in order to deal with the heterogeneity of mobile devices with different battery capacities (shown in Table 4.2), we use the percentage battery drain to make allocation decisions.

In order to optimize allocation decisions, it is important to get a good estimate of the instantaneous power drawn in the mobile device while running a workload task. This is

Table 4.3: The five workloads used for the evaluation of our proposed uncertainty-aware management framework.

Workflow	**No. of stages**	**No. of tasks**	**No. of task types**
1	4	20	10
2	6	30	10
3	8	40	10
4	11	60	15
5	13	70	15

obtained as follows: we ran the different workload tasks of our biomedical applications on the individual mobile devices to determine the instantaneous power drawn (i.e., to determine the current drawn in mA as the voltage drop remains constant) and the total time taken for the task completion. The voltage values did not show significant variability. From these observations, we determined the average time taken to complete a task of "unit" size, i.e., time taken to process input data of size $10KB$.

The workflows: We used different workflows to evaluate the performance of the two phases of the proposed solution. The 4-stage biomedical workflow for stress detection (shown in Fig. 4.3) is used to illustrate the benefits of data-uncertainty handling. However, evaluation at scale necessitates the creation of arbitrary workflows that are similar to the ones available in literature. Hence, we developed a workflow generator to generate arbitrary workflows that are inspired by the applications discussed earlier. The workflows used in our simulations vary in terms of the number of stages, number of tasks per stage, the types and sizes of tasks at each stage, and the dependencies as shown in Table 4.3. Our synthetic workflows are representative of a wide range of task-parallel applications that our solution can support and are used to compare the performance of our heuristic for multi-objective optimization with competing single-objective optimization approaches. The workflows are all task-parallel with data-parallel sub-graphs built into some of them.

4.4.1 Uncertainty vs. Non-uncertainty Awareness

To understand the benefits of data-uncertainty awareness, i.e., Phase I of our proposed solution, we studied the propagation of uncertainty in sensed data to the results of intermediate and final tasks in the workflow. The uncertainty in the sensed data is controlled by the

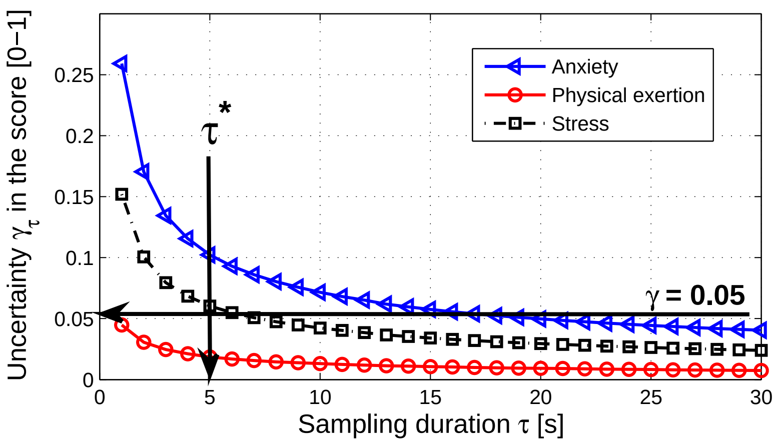


Figure 4.9: Decrease in uncertainty in the outputs of the tasks (in terms of width of the interval within which the score lies) with increase in τ. The pre-specified $\gamma = 0.05$ and the optimal sampling duration, which results in $\gamma_{\tau^*} \leq \gamma$ is $\tau^* = 5$ s.

sampling duration τ. As τ increases, the width of the 95% confidence interval of collected samples decreases, and so does the uncertainty in the score for HR, BP, GSR, and Activity, as shown in Fig. 4.5. The uncertainty in the output of Stage 1 of the workflow is propagated up the data-processing chain. Stage 2 of the 4-stage biomedical workflow is composed of simple computational models for anxiety-level and physical-exertion-level ratings. Anxiety score is a *weighted sum* of HR, BP, and GSR scores; while the physical-exertion score is the *product* of HR and Activity scores. At the final Stage 3, the stress score is a *weighted sum* of anxiety and physical-exertion scores.

To verify the performance of our algorithm for the C-fusion Case, the rules of interval arithmetic (shown in Table 4.1) were applied to propagate the uncertainty in the scores from Stage 1 to the final result in Stage 3. Let the uncertainty in the final result for a particular τ be γ_τ. The sampling duration τ^* for which the uncertainty in the final result γ_{τ^*} is not higher than the pre-specified γ, i.e., $\gamma_{\tau^*} \leq \gamma$, is the optimal sampling duration, which we find using linear search. A longer sampling duration results in unnecessary energy expenditure for processing additional data, while a shorter duration results in higher uncertainty in the result, as shown in Fig. 4.9 where $\gamma = 0.05$ and $\tau^* = 5$ s. Note that a $\tau > 5$ s does not decrease the uncertainty greatly, while a $\tau < 5$ s fails to meet the pre-specified $\gamma = 0.05$. Uncertainty awareness helps determine the most appropriate task sizes for the workflow tasks (based on τ^*), which not only has an effect on the accuracy but also on the timeliness

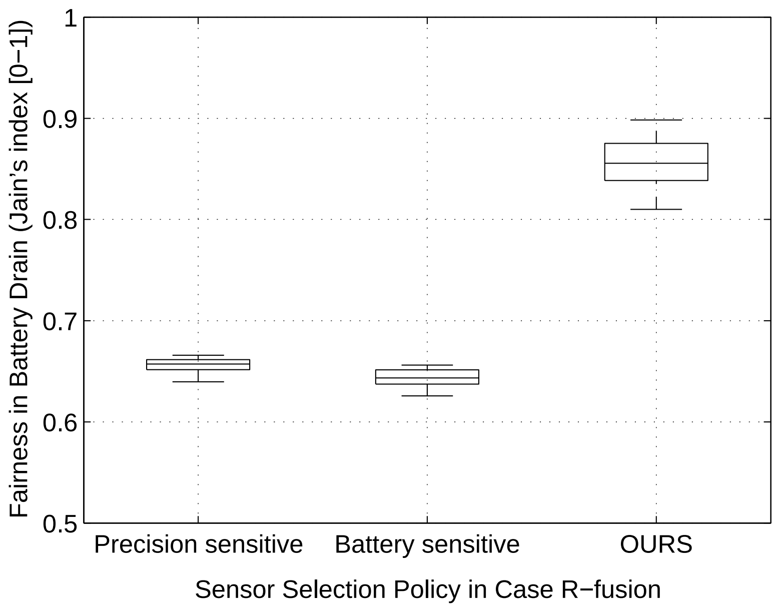


Figure 4.10: Algorithm for determining data quality and quantity in the R-fusion case ensures fairness in battery drain across all data providers. Our algorithm outperforms naive policies that are sensitive to sensor precision and to battery drain.

of the final result. Note that γ is just a requester-specified constraint on the quality of the result and has no effect on the performance of the proposed solutions.

In order to evaluate the performance our algorithm for the R-fusion Case, we extend the previous experiment scenario to include multiple data providers for HR (ECG *and* wrist-located pulse detector) and Activity (accelerometer *and* gyroscope). The sensors differ in terms of the precision of their outputs and battery usage profile. Figure 4.10 compares the performance of our algorithm for determining data quality and quantity against naive sensor selection policies. The metric we use is the Jain's fairness measure for the battery drain across all data providers. In Fig. 4.10, it can be seen that our sensor selection policy in the R-fusion Case outperforms naive precision- and battery-sensitive policies by a significant margin. The experiment was repeated several times under different initial conditions (initial battery levels) for statistical relevance. The horizontal line in the box plot (in Fig. 4.10) is the median, the bottom and top of the box represent the first and third quartiles, respectively, and the whiskers represent the minimum and maximum values.

4.4.2 Performance of `Fast M-CBP` Algorithm

Competing strategies: We assessed the performance of our heuristic for multi-objective optimization by comparing it against two popularly-used single-objective approaches: the first one, called *MinMax(Battery)*, tries to achieve fairness in battery drain; whereas the second one, called *MinMax(Time)*, tries to minimize the overall makespan of the mobile-application workflows by minimizing the time taken at every stage of the workflow. Note that, as the information regarding the underlying resource pool is unavailable to the end-user application, constraints cannot be provided to the single-objective approaches. Also, even conservative guesses for constraints may result in infeasibility. Five different workflows (shown in Table 4.3) differing in terms of number of stages, tasks, and heterogeneity of tasks were used to analyze the performance of all the approaches. As a single run of the workflow does not result in significant battery drain, the workflows were executed consecutively 200 times. Results presented are averages over 40 such runs (with different initial conditions in terms of battery capacity of the SPs) for statistical significance. The metrics of interest are makespan, fairness in battery drain, total battery drain, and the amount of data injected into the network.

Observations: Figure 4.11 shows that our heuristic for multi-objective optimization outperforms MinMax(Battery) by 15% *in terms of makespan*, but is outperformed by MinMax(Time). Conversely, as shown in Fig. 4.12, our heuristic outperforms MinMax(Time) by 56% *in terms of fairness*, but is outperformed by MinMax(Battery). From these two figures, it is clear that while MinMax(Time) and MinMax(Battery) perform very well in terms of their corresponding objective metric (i.e., makespan and fairness in battery drain, respectively), their performance in terms of the other metric is *very poor.* On the contrary, our multi-objective approach *finds a balance* in terms of both metrics. Figure 4.13 shows the total battery drain at all the SPs. MinMax(Time) seems to have incurred the least total battery drain; however, as already seen in Fig. 4.12, this single-objective approach drains the fastest and the most efficient SP's batteries as it does not concern itself with the longevity of the entire resource pool. Our multi-objective optimization heuristic incurs a similar total battery drain as the energy-efficient MinMax(Battery). Finally, Fig. 4.14

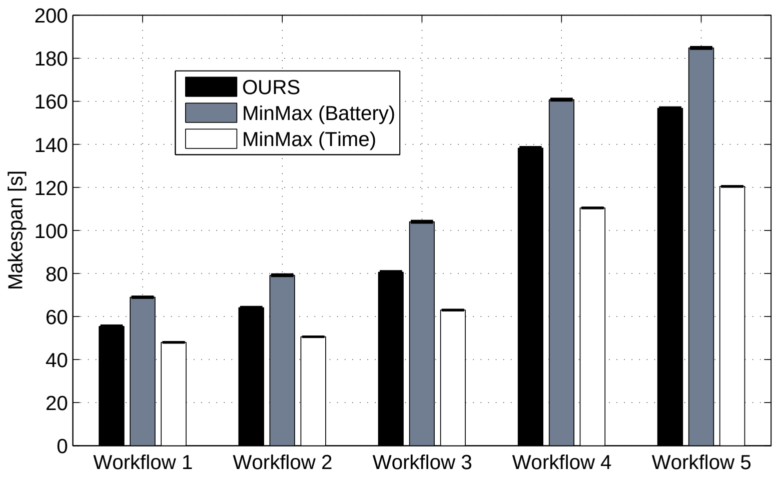


Figure 4.11: Average makespan [s] of five different workflows achieved by the three competing task-allocation mechanisms.

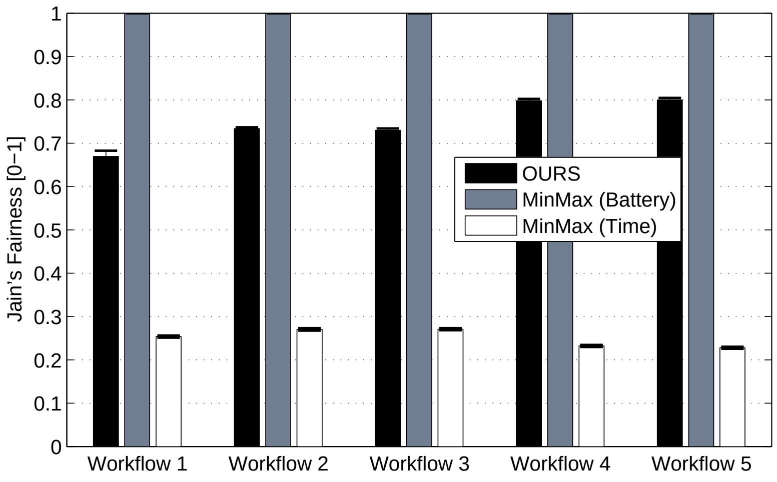


Figure 4.12: Average Jain's fairness (ranging in $[0, 1]$) in battery drain achieved by the three competing task-allocation mechanisms.

shows the network load (in terms of kilo Bytes injected into the network) incurred by the three approaches: our heuristic incurs only slightly more load than MinMax(Battery) while significantly outperforming MinMax(Time). This is because in MinMax(Time) most of the tasks are allocated to the fastest device, which results in significant network load.

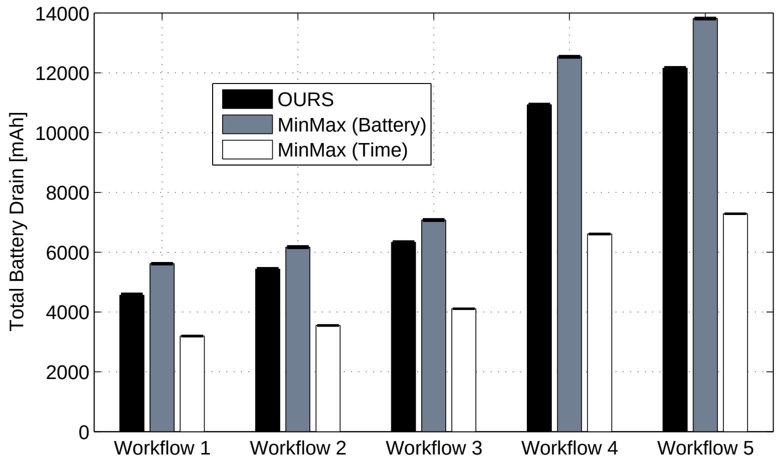


Figure 4.13: Total battery drain [mAh] achieved by the three competing task-allocation mechanisms.

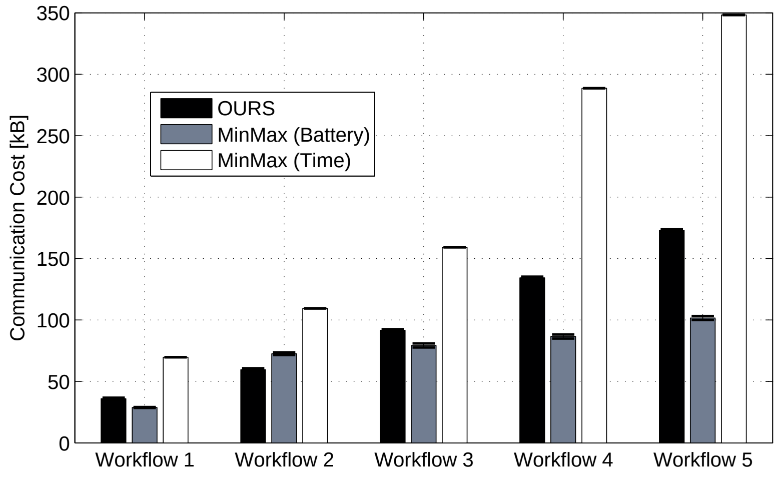


Figure 4.14: Average communication cost or network load [KB] (injected into the network for a *single* run of five different workflows) incurred by the three competing task-allocation mechanisms.

Chapter 5

Robust, Secure, and Privacy-preserving Mobile Grid Computing

5.1 Overview

In this chapter, we describe `Maestro`, a novel framework for robust, secure, and privacy-preserving mobile grid computing. In `Maestro`, any mobile application is represented as a workflow. Without any loss in generality, we assume that the role of arbitrator is played by one of the proximal fixed resource that is tethered to the Wi-Fi access point or the base station in order to ensure that all SPs are connected to arbitrator when they are in the network. Often, multiple service requests are received *simultaneously* by arbitrators and, hence, tasks belonging to multiple workflows managed by the arbitrators have to be allocated and executed on the SPs in the Cumulus. The aforementioned complex task-allocation problem, however, also presents opportunities: there may be multiple duplicate service requests at a arbitrator. Similarly, there may be multiple duplicate tasks that are common across different workflows.

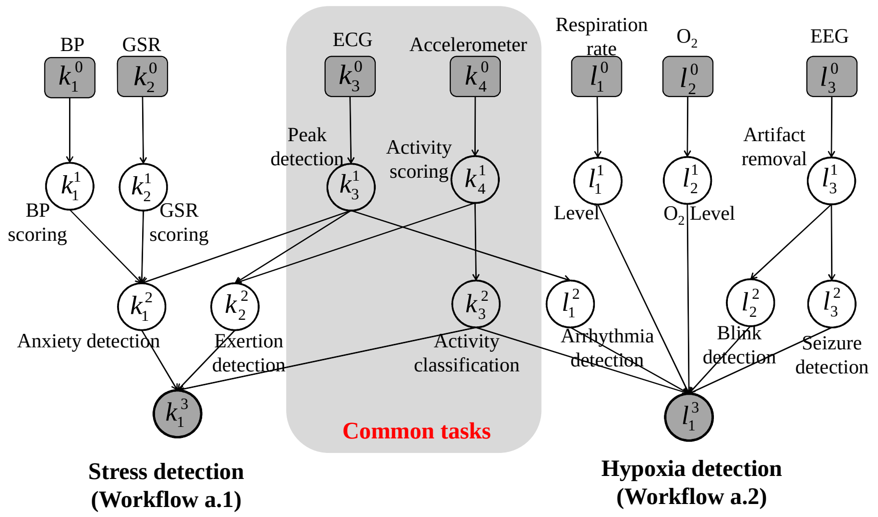
`Maestro` *deduplicates similar tasks across workflows* as it lends itself to minimization of duplication in services rendered. Task deduplication leads to efficient real-time in-situ processing of simplified workflows (with fewer tasks than before) as well as to better utilization of computing resources. To address the non-trivial research challenge of identification of task duplicates across workflows and the creation of simplified workflows (at the arbitrators), we introduce `Dedup`, a sub-graph matching technique for task deduplication among DAGs. After deduplication, the tasks of the simplified workflows have to be scheduled for execution on the Cumulus resources in such a way that the user-specified deadline of the corresponding workflow is met.

Aside from the simple yet effective reallocation of failed tasks, the scheduling mechanism in Maestro employs *controlled replication of critical workflow tasks* and *controlled access to sensitive user data* (via multiple levels of authorization) to realize *secure* and *privacy-preserving computing* in Cumulus, respectively. Controlled replication refers to the idea of replicating "critical" workflow tasks and only when needed the most (based on SPs' reliability). Task replication not only imparts *robustness* (against SP failures) but also provides *security* (from malicious nodes masquerading as SPs). Therefore, Maestro's task scheduling is *Byzantine fault tolerant* [58], i.e., it has the ability to handle *uncertainty* arising from device failures, denial of service, and intentional corruption of results. The issue of privacy is addressed by categorizing workflow tasks according to the sensitivity of the data processed and by allowing authorized service providers access only to appropriate task categories.

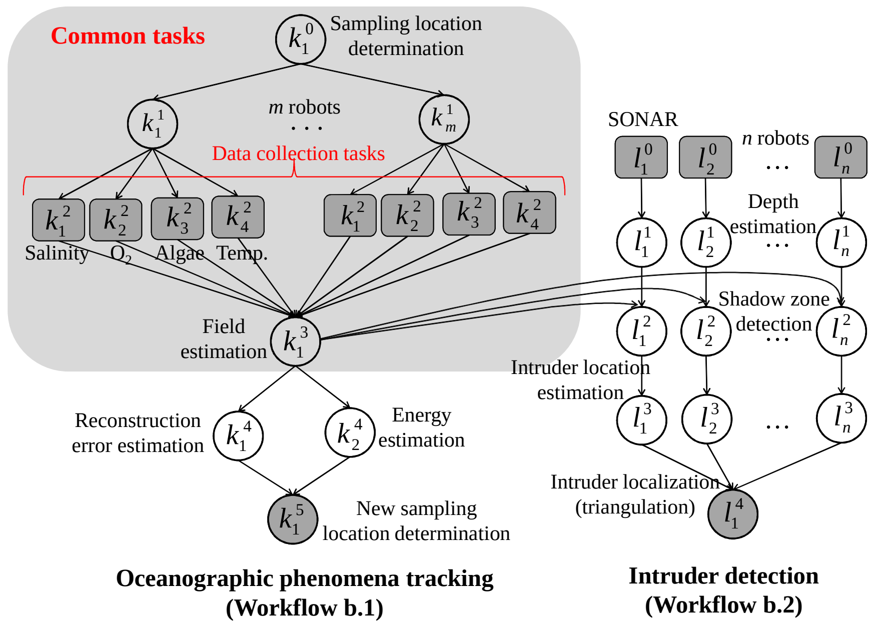
5.2 Concurrent Workflows

Often concurrent service requests will be received by each arbitrator, i.e., multiple workflows have to be executed concurrently in the underlying pool at any point in time. The aforementioned task allocation problem, albeit complex, presents opportunities. There may be multiple similar service requests at a arbitrator as well as multiple tasks that are common across different workflows. Deduplication of such common tasks leads to efficient real-time insitu processing of simplified workflows (with fewer tasks than before) as well as to better resource utilization. To understand the concept of concurrent workflows consider the following example applications from two different domains.

Ubiquitous healthcare: Applications in this domain (also called data-driven sensor-based healthcare) include stress detection, hypoxia (lack of oxygen) detection, alertness and cognitive performance assessment. Figure 5.1(a) depicts the task-parallel workflows for stress detection (Workflow a.1) and hypoxia detection (Workflow a.2), which use vital-sign data acquired from biomedical (e.g., BP, ECG, EEG) as well as kinematic sensors (e.g., accelerometer, gyroscope) attached to a person. Dummy tasks are not shown for the sake of simplicity. The tasks in these two workflows belong to one of the following classes: data analysis, data manipulation, decision making; and they aid in determining the psychophysiological state of a person (knowledge) from raw sensor data. As mentioned



(a) Ubiquitous healthcare domain


Common tasks
Sampling location determination
m robots
Data collection tasks
Salinity
O$_2$
Algae
Temp.
Field estimation
SONAR
n robots
Depth estimation
Shadow zone detection
Intruder location estimation
Reconstruction error estimation
Energy estimation
New sampling location determination
Intruder localization (triangulation)
Oceanographic phenomena tracking (Workflow b.1)
Intruder detection (Workflow b.2)

(b) Distributed robotics domain

Figure 5.1: Example application workflows from two different domains. The common tasks across workflows in the same domain are highlighted. Workflows in (a) are purely task-parallel while the ones in (b) are mixed.

earlier, applications belonging to the same domain may have the similar component tasks that can be deduplicated. For example, feature extraction from accelerometer outputs as well as ECG analysis (in Stage 1) are component tasks in hypoxia detection workflow (for context assessment, e.g., activity and arrythmia detection in Stage 2) as well as in stress-detection workflow (for exertion detection in Stage 2) as shown in Fig. 5.1(a).

Distributed robotics: Distributed decision-making applications in this domain include adaptive sampling, intruder detection, target tracking, data-driven path/trajectory planning. Figure 5.1(b) depicts the workflows for adaptive monitoring of aquatic ecosystem (Workflow b.1) and coastal underwater intruder localization (Workflow b.2), which use all or a subset of the following data acquired using environmental and inertial navigation sensors as well as SONAR on autonomous underwater robots: temperature, salinity, pollutants, nutrients, position, and depth. Workflow b.1 depicts how field estimation is used to track the effect of oceanographic phenomena (e.g., temperature and salinity gradients, algae growth, nutrient concentration) on aquatic life. Workflow b.2 depicts intruder localization (using SONAR) which requires optimal positioning of the robots in order to avoid false positives due to severe transmission losses in the acoustic signal in certain regions. Such regions of high transmission loss can again be determined from temperature, salinity, and depth field estimates. Here, field estimation is a common task (between the two workflows), which can be deduplicated. Note that in this application the overall task-parallel workflow is composed of smaller data-parallel workflows.

5.3 Maestro

In this section, we present Maestro, a robust, secure, and privacy-preserving mobile grid computing framework for concurrent workflow management on Cumuli. Firstly, we present Dedup (the sub-graph matching technique in Maestro) for task deduplication among DAGs. Secondly, we discuss the Maestro's task scheduling mechanism, which employs controlled task replication (for robustness and security) before scheduling the tasks for execution on appropriate Cumulus resources.

5.3.1 Task Deduplication

The arbitrators receive multiple workflow execution requests over a period of time from the service requesters. The arbitrators group service requests before proceeding with task

Algorithm 5 `Dedup`

Input: $\mathcal{K}$ and $\mathcal{L}$ are initially set to stage-0 tasks of the two workflows
Output: Simplified workflows

```
for every $k \in \mathcal{K}$ do
  if visited($k$) == true then
    continue
  end if
  for every $l \in \mathcal{L}$ do
    if visited($l$) == true then
      continue
    end if
    if checkSimilarity($l$,$k$) == true then
      Dedup($\mathcal{C}^k$,$\mathcal{C}^l$)
    else
      Parent($l$) = Parent($k$)
      addChild(Parent($k$),$l$)
    end if
  end for
end for
```

Algorithm 6 `checkSimilarity`

Input: Tasks k and l
Output: `true` or `false`

```
if $k.taskID == l.taskID$ AND $k.output == l.output$ then
  if checkSimilarity($\mathcal{P}^k$,$\mathcal{P}^l$) then
    visited($k$) = visited($l$) = true
    visited($\mathcal{P}^k$) = visited($\mathcal{P}^l$) = true
    return true
  else
    return false
  end if
else
  return false
end if
```

deduplication. The duration (time window) for which a arbitrator waits (δ_b^{wait}) before deduplication is a tunable parameter. For a given rate of service request arrivals, the larger the window the greater the chances of finding task duplicates. However, the windows cannot be too large as the workflow requests have to be serviced real time. This "pause-aggregate-service" strategy eliminates the simplifying assumption of strictly simultaneous workflow arrivals at the arbitrators.

`Dedup` – at the arbitrator – parses the workflow descriptions to identify task duplicates and to create simplified workflows (with fewer tasks than before). `Dedup` looks for matching subgraphs (connected group of tasks) between a pair of DAGs. Trivially, every single vertex in a DAG (workflow) is a subgraph. `Dedup` starts with the comparison of stage 0 tasks in the two workflows as shown in Algorithm 5. Two tasks are considered to be "similar" when the following attributes match: task identifier (i.e., type), number and types of inputs (i.e., set of parent tasks, $\mathcal{P}$), and inputs' sizes (quantity of inputs) as shown in Algorithm 6. When tasks in two DAGs are similar, their corresponding sets of child tasks ($\mathcal{C}$s) are recursively checked for similarity. This recursive step is aimed at growing the size (i.e., number of tasks) of the matched subgraph. In the recursive procedure, when the tasks under comparison, say task k of workflow 1 and l of workflow 2, cease to be similar, a link is created from k's parent to l. Also, l is added to the children set of Parent(k). The tasks belonging to the duplicate subgraph in workflow 2 are discarded. Note that while checking for similarity, tasks that have been visited and have tested positive for similarity are marked so that they need not be checked again. The worst-case time complexity of `Dedup` is $\mathcal{O}(|\mathcal{V}_1| \cdot |\mathcal{V}_2|)$, where $\mathcal{V}_1$ and $\mathcal{V}_2$ are the sets of vertices in the two input DAGs.

In the resulting workflow, k's immediate predecessor task will be "fork" point from which other deduplicated workflow branches out. In Fig. 5.1(a), the vertices corresponding to peak detection in ECG signal and to PSD analysis of accelerometer output become fork points. Note that the time complexity of `Dedup` (or the total number of comparisons) is not altered by altering the order of comparison of different DAGs. Similarly, `Dedup` results in the same set of simplified workflows irrespective of the order of comparison of the different DAGs. It is not necessary that deduplication has to be done strictly before task allocation. Deduplication is also achieved on the fly when workflow tasks are already in execution. Under such circumstances, execution of duplicates is piggybacked thus deduplicating at run time. Also, results of certain repetitive workflow tasks are cached locally at SPs so to deduplicate services.

5.3.2 Concurrent Workflows Scheduling

After deduplication, the simplified workflows arrive at the arbitrator. The tasks of these workflows have to be allocated to SPs in the Cumulus. While submitting a service request, the application can specify the absolute deadline (D) within which the workflow execution has to be completed for it to be useful. There is also a notion of an acceptable probability of failure (P^{fail}) for each workflow. This probability can be a service level guarantee advertised by the arbitrator or negotiated a priori between arbitrators and service requesters. `Maestro`'s task scheduling mechanism at the arbitrator is in charge of determining i) the set of workflow tasks that are ready to be allocated, ii) the relative priority among the ready tasks[1], iii) the amount of replication and the appropriate SP(s) for each ready task.

In `Maestro`, tasks can be immediately allocated as and when they become ready or the ready tasks can be accumulated (over a waiting period, δ_b^{ready}) and then allocated for a more efficient schedule in terms of makespan (total workflow execution time) and number of replicas (i.e., battery drain). This waiting period is again a tunable parameter. The larger the waiting period, the greater the chances of finding the most appropriate SPs (lower makespan and fewer replicas). However, δ_b^{ready} cannot be too large due to real-time constraints of the application.

Task prioritization: Determining the relative priority among ready tasks from the same or different workflows requires incorporation of computation-time information and deadline requirements as discussed in prior work on workflows management in computational grids [38]. Firstly, we determine the *level* of task, which is the length of the longest path from that task to an exit task. The length of a path in a DAG is the sum of the average computation time of that task and the average computation times of all the successor tasks along the path. The average communication times between successive tasks should also be taken into account if the Communication to Computation costs Ratio (CCR) of the workflow DAG is high. The level (Δ[s]) of a task k is given by,

$$\Delta^k = \overline{\alpha^k} + \max_{c \in \mathcal{C}^k}\{\overline{\beta^{kc}} + \Delta^c\}, \tag{5.1}$$

where $\overline{\alpha^k}$ is the average computation time of a task k on the SPs in the Cumulus, $\mathcal{C}^k$ is the set of child tasks of k, and $\overline{\beta^{kc}}$ is the average communication time for data transfer between tasks k and c when executed on the SPs in the Cumulus.

Once the level of each ready task is known, their *slack* S (maximum allowable wait time

[1]A ready task is one that does not have any unresolved dependencies, i.e., all its parent tasks have completed execution.

before execution of that task) at any time t is determined as,

$$S^k(t) = D^k - \Delta^k - t, \tag{5.2}$$

where D^k is the absolute deadline for the workflow, which task k belongs to. The task with the *smallest* slack S has the *highest* priority. After prioritization of the ready tasks according to this criteria, the most appropriate SP for allocation and the amount of replication (when necessary) are determined. Each service provider in the Cumulus has a task queue. For a ready task k with the highest priority, the SP n that provides the earliest finish time ($t_n^{k,fin}$) is the most preferred. Finish times are considered due to heterogeneity in capabilities of service providers. In a homogeneous environment, start times are sufficient to make allocation decisions. The t^{fin}s are obtained as,

$$t_n^{k,fin} = t_n^{k,start} + \alpha_n^k, \tag{5.3}$$

where $t^{k,start}$ is the start time for task k on SP n. The t^{start} depends on the number and type of existing tasks in the task queue. However, there is uncertainty associated with the availability of the SPs in a Cumulus for the required duration and this has to be taken into account in the scheduling mechanism.

Controlled replication: An effective way to overcome the uncertainty (due to failures) is reallocation of failed tasks (also called "healing"). However, healing is not suited for tasks with large computation times and tasks that are critical for multiple workflows after deduplication. Though healing provides robustness it increases the makespan as it waits for at least the task's computation time before making a decision (*reactive*). Hence, we replicate critical tasks at multiple service providers (*proactively*) to ensure the completion of those tasks on time. Proactive task replication avoids unnecessary idle waiting times incurred in reactive failure handling (i.e., healing). Tasks that have to be replicated are allocated to the SP that provides the next earliest $t^{k,fin}$. Note that, as replicas have the same priority as the original, they are allocated together with the original before the other tasks that have lower priority.

All tasks in a workflow should not be replicated as it will increase the total number of tasks to be executed in turn leading to massive queuing delays and large makespans. The application developer can explicitly annotate certain workflow tasks as non-critical. All other tasks are treated as *blocking tasks*, i.e., the progress of the workflow depends on the completion of these tasks. The decision to replicate a task k initially allocated to SP n is taken based on how the task-completion probability of n compares with the "required" success probability for that task derived from the pre-specified P^{fail}. The required success probability p^{succ} for each task in the set of incomplete tasks $\mathcal{K}$ of a workflow is obtained by solving,

$$(p^{succ})^{|\mathcal{K}|} = 1 - P^{fail}. \tag{5.4}$$

The task-completion probability $p_n^{k,succ}$ of a task k at SP n is defined as,

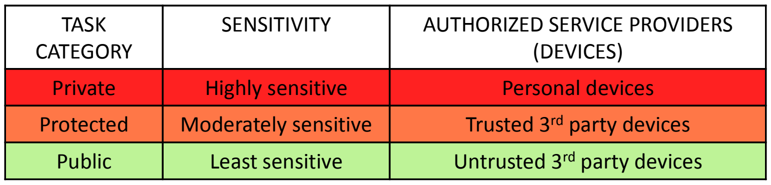
$$p_n^{k,succ} = Pr\{t + T^n > t_n^{k,start} + \alpha_n^k\}, \tag{5.5}$$

where t is the current time and T^n is the "actual" availability duration of SP n. Without any loss in generality, we assume that the distribution of service provider availability duration is known while determining $p_n^{k,succ}$. It is pretty straightforward to obtain and maintain this statistic at the arbitrators. When $p_n^{k,succ} < p^{succ}$, a replica is allocated to the next best SP as mentioned before. Replicas of task k are created and allocated to SPs until the following condition is satisfied for the first time

$$1 - \prod_{n \in \mathcal{N}} (1 - p_n^{k,succ}) \geq p^{succ}, \tag{5.6}$$

where $\mathcal{N}$ is the set of SPs to which the replicas are allocated. Note that, as the tasks of a workflow are completed over time with probability one, the required success probability of remaining incomplete tasks decreases. This allows the scheduler to use some less reliable SPs thus resulting in load balancing.

For "fork" tasks that are common across multiple workflows, the more stringent condition on the required probability of success is taken into account. To avoid uncontrolled replication, we use a maximum replication limit. This limit is different for different types



TASK CATEGORY	SENSITIVITY	AUTHORIZED SERVICE PROVIDERS (DEVICES)
Private	Highly sensitive	Personal devices
Protected	Moderately sensitive	Trusted 3rd party devices
Public	Least sensitive	Untrusted 3rd party devices

Figure 5.2: The three categories of tasks and the authorized service providers that can perform the different categories of tasks.

of tasks. For example, the fork tasks, which are crucial for the success of multiple workflows have a greater replication factor than the other tasks. This difference in the level of protection is crucial to avoid blocking of Cumulus resource by tasks that are not so critical as the ones that follow them.

Byzantine fault tolerance: Apart from imparting robustness, task replication also enables identification of malicious mobile devices in the network. This can be achieved through comparative analysis of outputs of task replicas from multiple SPs. Malicious devices, masquerading as SPs, may tamper with the data and intentionally generate inaccurate results to adversely affect the workflow execution. `Maestro` is designed to be Byzantine fault tolerant. i.e., it is capable of handling arbitrary failures not only due to SP failures or loss in SP connectivity but also due to inconsistent behavior by malicious devices masquerading as SPs. Presently, we focus only on the cases where the malicious devices do not provide any result at all (service denial).

5.3.3 Privacy-preserving Computing

Privacy is a major concern in distributed processing of "personal sensor data" (e.g., vital-signs, location) on geographically proximal volunteered computing resources. We address this issue by assigning different levels of protection to the different categories of tasks, i.e., we determine what computing resources the different tasks are assigned to. We elaborate on our idea under the context of a ubiquitous health monitoring application. Data-analysis tasks are basic statistical methods that run over a tremendous amount of time-series data. The knowledge of the "data type" and context is inconsequential for the data-analysis tasks and, hence, the data can be anonymized so to not provide any private information (e.g., participant's identity and health status). Therefore, data-analysis tasks (*public* tasks) can be performed on *any* "volunteered resource" in proximity without any concerns over the level of trust of the computing resource.

Conversely, data-manipulation tasks (e.g., artifact removal in biomedical signals) need to be aware of the data type and, hence, can be carried out *only* on "trusted resources". However, they do not need any contextual information or identity of the participant whom the data belongs to. Trusted resources include service providers belonging to family members and friends (on social networks and real life) and this category of tasks is referred to as *protected* tasks. Differently from the other two, decision-making tasks require the participant's identity and contextual information to generate baseline information (e.g, health-status of participants in a biomedical application). Therefore, these *private* tasks can *only* be performed on the participant's "personal mobile devices" (highest level of trust). The aforementioned restrictions arising out of privacy concerns are taken into account in the task scheduling mechanism of `Maestro` in the form of constraints.

5.4 Evaluation

We developed a simulator in Java^{TM} to empirically evaluate the performance gains provided by different components of `Maestro`. Simulations also allow us to evaluate at scale. In the following, firstly, we present details about our experiment methodology, specifically, the workflows, workflow traces, the service providers, and the SP dynamics. Then, we discuss specific simulation scenarios and the results that i) demonstrate the benefits of `Dedup`, ii) highlight the price of privacy preserving mobile computing, and that iii) illustrate the merits of replication-based failure handling.

5.4.1 Methodology

Workflows: The workflows we used in our simulations are inspired by the applications discussed earlier. The workflows are all task-parallel with data-parallel sub-graphs built into some of them. Even though currently available example application workflows (from the biomedical and distributed robotics domains) can be used for preliminary evaluation of `Maestro`, evaluation at scales necessitates the creation of arbitrary workflows that are similar to the ones available in literature. Hence, we developed a workflow (DAG) generator to generate arbitrary workflows for large simulations. The workflows used in our simulations vary in terms of the number of stages, number of tasks per stage, the types and sizes of tasks at each stage, and the dependencies. Our synthetic workflows are representative of a wide range of task-parallel applications that `Maestro` can support.

Workflow trace: At different points in time, the service requesters (which may also be data providers) submit workflow requests to arbitrators while also specifying a deadline and a probability of success. The utility of the result from the workflow is assumed to be zero after the requester-specified deadline. Traces that capture workflow request arrivals over time in a mobile computing environment are not available in literature. Workflow arrival traces in cloud and grid computing environments cannot be adopted directly either. This is because the workflows, their deadline requirements, and arrival statistics are not representative of the applications or of the dynamics envisioned in `Maestro` or any other work on mobile device clouds and opportunistic computing. We developed a workflow trace generator, which can create multiple workflow arrival traces that vary in terms of the number of workflows, inter-arrival time between workflows as well as the request-specific deadline and probability of success.

Service providers: In our simulations we use a heterogeneous pool of SPs. The factors that contribute to heterogeneity are processing speed or capability (in terms of number of instructions per second), communication capability (in terms of bps), rate of battery drain for computation (in terms of mAh per instruction) and communication (in terms of mAh per second), and finally the duration of availability. We use the mean availability duration (the duration for which the SP is in the Cumulus) and the mean away duration (the duration for which the SP is not in Cumulus) as well as their respective distributions to control the dynamics in the mobile computing environment. We choose these durations carefully to maintain an average number of SPs in the Cumulus as the three variables are related by Little's theorem.

5.4.2 Benefits of Task Deduplication

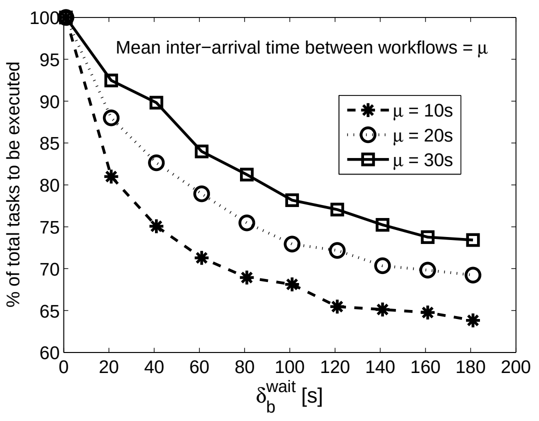
In order to demonstrate the benefits of task deduplication, we did two experiments to ascertain the following two factors with and without Dedup: 1) the reduction in percentage of total number of tasks to be executed in the Cumulus and 2) the percentage of successfully completed workflows among all the workflow requests submitted.

Experiment 1: As δ_b^{wait} is a tunable parameter, we observed performance in terms of reduction in number of tasks by varying it. We studied the behavior of Dedup when δ_b^{wait} is adapted to the arrival rate of workflows and when it is not. We created three different workflow traces each with a total of 100 requests. The mean inter-workflow-arrival durations in the three workflow traces are $\mu = 10, 20$, and $30s$, respectively. The number of distinct workflows in each trace was set to 40 and the number of SPs to 10.

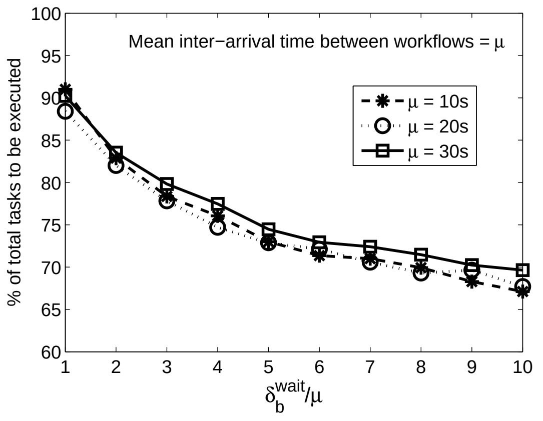
Observations: Figures 5.3(a) and 5.3(b) show that the percentage of total number of tasks to be executed in the Cumulus decreases by 25% when δ_b^{wait} is five times the inter-arrival duration μ of the workflows. This decrease will be greater when the number of distinct workflows in the traces is reduced below the current value of 40. Figure 5.3(a) was obtained by varying δ_b^{wait} in increments of 20 agnostic to the workflow arrival rate. As a result in comparison to the trace with $\mu = 20s$, the percentage of tasks to be executed is higher for the trace with $\mu = 30s$ while it is lower for the one with $\mu = 10s$. This is because for the same δ_b^{wait}, the number of workflows considered together for deduplication decreases with increase in μ. Therefore, adaptation of δ_b^{wait} with respect to μ is key to achieve improved performance as shown in Fig. 5.3(b).

Experiment 2: We observed the performance in terms of percentage of successful workflow completions by varying the waiting period δ_b^{wait}. We created a workflow trace with a total of 500 requests. The mean inter-workflow-arrival duration in the workflow trace was set to $\mu = 10s$ and the number of distinct workflows in the trace was set to 10. The deadline of each workflow request was chosen randomly between 40 and $80s$ and the number of SPs to 10.

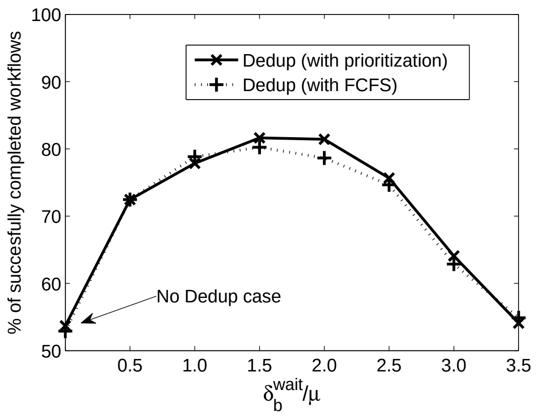
Observations: Figures 5.3(c) shows that the percentage of successful workflow completions in the Cumulus increases to as much as 83% (compared to the baseline no Dedup case



(a)



(b)



(c)

Figure 5.3: Decrease in percentage of the total number of tasks to be executed (while using `Dedup`) with increase in δ_b^{wait} (a) when δ_b^{wait} is not adapted to the arrival rate of workflows; and (b) when δ_b^{wait} is adapted to the arrival rate of workflows (proportionally). (c) Behavior of `Dedup` in terms of percentage of successfully completed workflows with increase in δ_b^{wait}.

at 53%) when δ_b^{wait} is twice the inter-arrival duration μ of the workflows. This increase will be greater when the failed workflow tasks are discarded (which we did not do to study the worst case). Figure 5.3(c) clearly highlights the situation when `Dedup` may not be beneficial in `Maestro`. Even though an increase in δ_b^{wait} results in a decrease in the total number of tasks to be executed as shown in Figs. 5.3(a) and 5.3(b), the decrease is only sub-linear. The accumulation of tasks over time may result in an overload for the underlying SP pool as is the case when $\delta_b^{wait}/\mu > 2.0$ in Fig. 5.3(c) where the gain drops until finally reaching the baseline at $\delta_b^{wait}/\mu = 3.5$. Also, it is important to note that the task prioritization in `Maestro` results in improved performance in comparison to a First-Come-First-Served

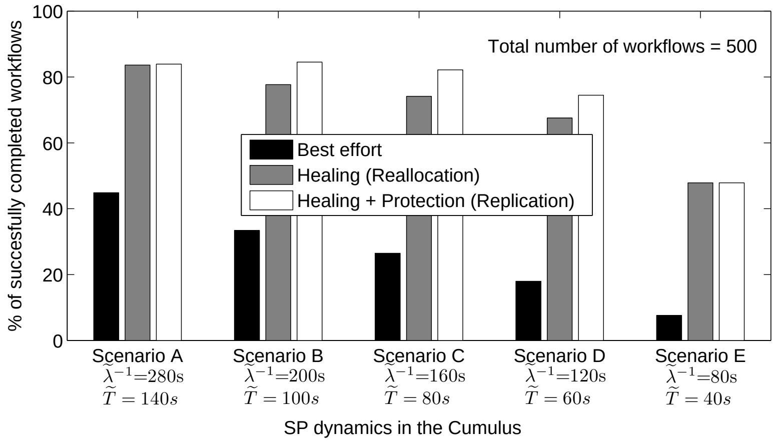


Figure 5.4: Percentage of successful workflow completions under different SP dynamics in the Cumulus. Scenarios A through E represent a progressive decrease in the stability of SPs. $\widetilde{\lambda}^{-1}$ [s] is the average inter-arrival duration and $\widetilde{T}$ [s] is the average availability duration of the SPs.

(FCFS) scheduling policy. The difference in performance will widen further when the variance in deadlines is greater than what we used here.

5.4.3 Benefits of Controlled Replication

Maestro's task scheduling mechanism employs proactive protection (selective controlled replication of large tasks) in addition to reactive healing (reallocation of failed tasks) in order to provide robustness. To study the improvement in performance provided by healing and protection over the best-effort (baseline) case, we performed an experiment under different service provider dynamics. We set the *average number of active SPs* in the Cumulus to 30. We varied the SP dynamics in the Cumulus from highly volatile to highly stable by tuning the following parameters - *average inter-arrival duration* ($\widetilde{\lambda}^{-1}$ [s]) of SPs and *average availability duration* ($\widetilde{T}$ [s]) of SPs as shown in Fig. 5.4. The average number of SPs and the aforementioned parameters are related by Little's law. We created a workflow trace with a total of 500 requests. The mean inter-workflow-arrival duration in the trace was set to $\mu = 20s$ and the number of distinct workflows was set to 10. The trace has a mix of small (66%) and large (33%) workflows (differing in terms of task sizes and deadlines). The deadline of the small workflows was chosen randomly between 40 and $80s$ while the large workflows' deadline was picked randomly between 80 and $160s$.

Observations: Figure 5.4 shows the five scenarios A through E, where A corresponds to a highly stable Cumulus and E corresponds to a highly volatile one. The performance of Maestro with only healing and with both healing and protection is always better than the baseline case. In scenarios B through D the use of protection in addition to healing prevents more workflows from failing than the use of plain healing. This is because while using healing in isolation, the time taken to recover from a failure of a task belonging to a "large" workflow is more than twice that task's execution time. However, selective replication of such critical tasks (which may easily jeopardize the workflow when they fail) as done in protection prevents a greater percentage of workflows from failing. However, note that protection does not provide any additional gain over plain healing in Scenario A when probability of SP failure during task execution is very low and in Scenario E when probability is very high.

5.4.4 Price of Privacy-preserving Computing

Even though Maestro provides different levels of protection to different tasks through controlled access by authorized service providers, there is a price to pay for privacy-preserving computing in terms of performance. Authorizing service providers to execute only certain types of tasks restricts the feasibility region of the solution to the problem the arbitrator is trying to solve. In Maestro, the arbitrator aims at scheduling tasks in the Cumulus in such a way that the percentage of successfully completed workflows is maximized. We performed an experiment to quantify the difference in performance when different privacy policies are employed. We varied the percentage of private, protected, and public tasks in the workflows while keeping the percentage of personal, trusted and untrusted 3^{rd} party devices in the Cumulus fixed. For any service request the percentage of personal, trusted, and untrusted devices are 1%, 33%, and 66%. We created a workflow trace with a total of 500 requests. The mean inter-workflow-arrival duration in the workflow trace was set to $\mu = 10s$ and the number of distinct workflows in the trace was set to 10. The deadline of each workflow request was chosen randomly between 40 and $80s$.

Observations: Figure 5.5 shows the 5 policies with decreasing degree of privacy (or increasing number of public tasks in the workflows). We observed that as the degree of

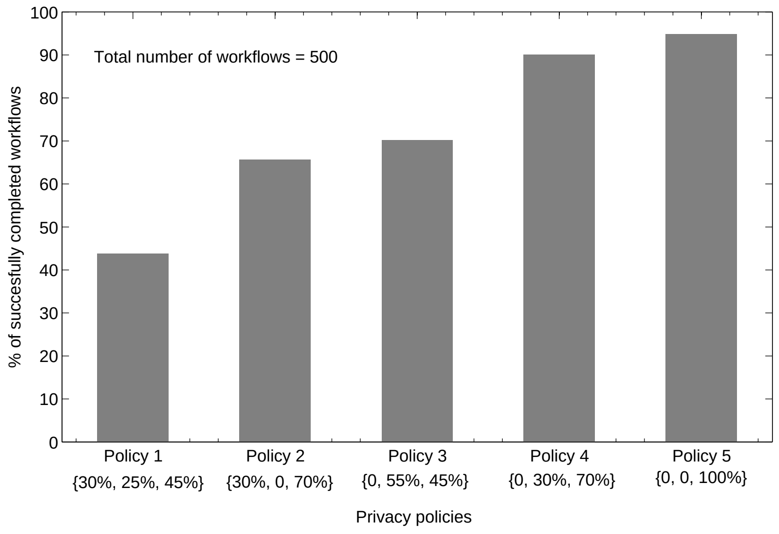


Figure 5.5: Percentage of successful workflow completions when different privacy policies are employed. The tuple represents the % of private, public, and protected tasks in each policy. Policies 1 through 5 represent a progressive decrease in the degree of privacy.

privacy is decreased, i.e., the percentage of public tasks increased, the performance in terms of percentage of workflow completions increased. Policy 5 represents an unrestricted scenario where all tasks are public while Policy 1 is extremely restricted. Policies 2 through 4 reflect what may be adopted in real-world deployments.The performance of Policies 2 and 3 can be improved to match that of Policy 4's by either relaxing the deadline requirements or by increasing the Cumulus size. A small relaxation in the deadline is a marginal cost to incur if privacy is desired.

Chapter 6

Conclusions and Future Research Directions

Tremendous advances in mobile computing coupled with the growing popularity of cloud computing have rendered the powerful mobile devices heavily under-utilized on average. In this dissertation, we presented our vision to *collectively* exploit the heterogeneous sensing, computing, communication, and storage capabilities of these under-utilized mobile devices in the field *as well as* that of computing and storage servers in remote datacenters in order to create a mobile computing grid (also referred to as a "loosely-coupled" mobile device cloud), called Cumulus. The Cumulus can be harnessed to enable novel data- and compute-intensive mobile applications that rely on *real-time in-situ processing* of data generated in the field.

We observed that the challenges facing real-time in-the-field data collection and processing using mobile platforms are: the inherent uncertainty associated with the quality and quantity of data from mobile sensors as well as with the availability and capabilities of mobile computing resources in the field, security, and privacy. Our goal was to design and develop a unified uncertainty-aware (robust), secure, and privacy-preserving framework for data and computing resource management in the Cumulus in order to enable execution of mobile application workflows in real time and in situ and, hence, to generate actionable knowledge from raw data within realistic time bounds.

In order to achieve the aforementioned goal, we proposed an autonomic (self-organizing, self-optimizing, and self-healing) middleware that aids in the organization of the sensing, computing, and communication capabilities of static and mobile devices in order to form Cumuli. As the relevance (level of accuracy and timeliness) of the output of workflows rely heavily on the quality and quantity of raw data coming from the underlying sensing infrastructure as well as the computing resources available to execute them in real time, we developed a unified data and computing resource management mechanism for data- as well as task-parallel applications. Finally, we proposed `Maestro`, a robust, secure, and privacy-preserving framework for concurrent mobile application management in Cumulus. We evaluated our contributions through experiments on a prototype testbed as well as through simulations on a purpose-built Java$^{\text{TM}}$-based simulator.

We have identified avenues for further research in the following areas:

Consensus-based applications: Even though the proposed solutions are capable of handling data-parallel workflows in which the result is generated by aggregating results from multiple computing resources, it is just a preliminary form of consensus-based application. Applications like joint localization, team formation, and collision avoidance in the robotics domain are examples of consensus-based workflows.

Byzantine-fault tolerance: Our research group at the Rutgers Cyber-Physical Systems (CPS) Lab is working towards addressing security issues that arise when malicious nodes provide incorrect results and not just denial of service attacks as it is done now in `Maestro`. This will make the framework "fully" Byzantine fault tolerant.

Heterogeneous mix of applications: As of the time of writing this dissertation our research group is involved is exhaustive validation of `Maestro` using a heterogeneous mix of mobile application workflows differing in terms of communication-to-computation-cost ratios and degree of concurrency.

Dynamic workflows: Model uncertainty has a significant effect on the relevance of the result and hence, fundamental research is required to construct or select the most appropriate workflow for real-time in-situ data processing in the Cumulus. On-the-fly generation of workflows that are tailored to the available data quantity and quality as well as to the available computing resources in the underlying heterogeneous multi-modal sensing and computing infrastructure is another potential avenue for further research.

References

[1] Volunteer Computing. http://www.gridcafe.org/volunteer-computing-.html?PHPSESSID=af0ca7e6d4ad7b5bc2f4ac314e190f95.

[2] Mahadev Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[3] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proc. of The European Professional Society on Computer Systems (EuroSys)*, Salzburg, Austria, April 2011.

[4] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. of the Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, June 2010.

[5] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: Enabling Interactive Perception Applications on Mobile Devices. In *Proc. of the Intl. Conf. on Mobile systems, Applications, and Aervices (MobiSys)*, Bethesda, MD, June 2011.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), October 2009.

[7] Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, Z. Morley Mao, and Xu Chen. COMET: Code Offload by Migrating Execution Transparently. In *Proc. of the USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, October 2012.

[8] Hariharasudhan Viswanathan, Eun Kyung Lee, Ivan Rodero, and Dario Pompili. An Autonomic Resource Provisioning Framework for Mobile Computing Grids. In *Proc. of the ACM Intl. Conference on Autonomic computing (ICAC)*, San Jose, CA, September 2012.

[9] Hariharasudhan Viswanathan, Eun Kyung Lee, and Dario Pompili. Mobile Grid Computing for Data- and Patient-centric Ubiquitous Healthcare. In *Proc. of IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT)*, Seoul, South Korea, June 2012.

[10] Hariharasudhan Viswanathan, Eun Kyung Lee, and Dario Pompili. Enabling Real-time In-situ Processing of Ubiquitous Mobile-Application Workflows. In *Proc. of the IEEE Intl. Conference on Mobile Adhoc and Sensor Systems (MASS)*, Hangzhou, China, October 2013.

[11] Hariharasudhan Viswanathan, Parul Pandey, and Dario Pompili. Maestro: Orchestrating Concurrent Workflows Execution in Mobile Computing Clouds. In *Submitted for peer review*, 2014.

[12] Hariharasudhan Viswanathan, Baozhi Chen, and Dario Pompili. Research Challenges in Computation, Communication, and Context Awareness for Ubiquitous Healthcare. *IEEE Communications Magazine*, 50(5):92–99, May 2012.

[13] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and Grid Technologies for Wide-Area Distributed Computing. *SOFTWARE: Practice and Experience*, 32(15):1437–1466, 2002.

[14] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications. In *Proc. of the ACM/IFIP/USENIX Intl. Conf. on Middleware (Middleware)*, Urbanna Champaign, IL, November 2009.

[15] Junseok Hwang and Praveen Aravamudham. Middleware Services for P2P Computing in Wireless Grid Networks. *IEEE Internet Computing*, 8(4):40–46, 2004.

[16] Jorg Roth and Claus Unger. Using Handheld Devices in Synchronous Collaborative Scenarios. *Ubiquitous Comp.*, 5(4):187–199, 2001.

[17] Yih-Farn Chen, Huale Huang, Rittwik Jana, Trevor Jim, Matti Hiltunen, Sam John, Serban Jora, Radhakrishnan Muthumanickam, and Bin Wei. iMobile EE–An Enterprise Mobile Service Platform. *Wireless Networks*, 9(4):283–297, 2003.

[18] Marco Conti and Mohan Kumar. Opportunities in Opportunistic Computing. *IEEE Computer*, 43(1):42–50, January 2010.

[19] Andrea Passarella, Mohan Kumar, Marco Conti, and Eleonora Borgia. Minimum-Delay Service Provisioning in Opportunistic Networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(8):1267–1275, August 2011.

[20] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proc. of the Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Hilton Head Island, SC, 2012.

[21] David Chu and Marty Humphrey. Mobile OGSI.NET: Grid Computing on Mobile Devices. In *Proc. of IEEE/ACM Intl. Workshop on Grid Computing*, Pittsburgh, PA, November 2004.

[22] Luciana dos S. Lima, Antônio T. A. Gomes, Artur Ziviani, Markus Endler, Luiz F. G. Soares, and Bruno Schulze. Peer-to-peer Resource Discovery in Mobile Grids. In *Proc. of the Intl. Workshop on Middleware for Grid Computing (MGC)*, Grenoble, France, November 2005.

[23] Sze-Wing Wong and Kam-Wing Ng. A Middleware Framework for Secure Mobile Grid Services. In *Proc. of IEEE Intl. Symp. on Cluster Computing and the Grid (CCGRID)*, Singapore, May 2006.

[24] Dijiang Huang, Xinwen Zhang, and Myong Kang andJim Luo. MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication. In *Proc. of IEEE Service Oriented System Engineering (SOSE)*, Nanjing, China, June 2010.

[25] Jarle Hulaas, Walter Binder, and Giovanna Di Marzo Serugendo. Enhancing Java Grid Computing Security with Resource Control. In *Proc. of Grid Services Engineering and Management(GSEM)*, Erfurt, Germany, September 2004.

[26] Rodrigo M. Barbosa and Alfredo Goldman. Mobigrid - Framework for Mobile Agents on Computer Grid Environments. In *In Mobility Aware Technologies and Applications (MATA)*, pages 147–157. Springer-Verlag, 2005.

[27] Paul J. Darby and Nian Feng Tzeng. Peer-to-peer Checkpointing Arrangement for Mobile Grid Computing Systems. In *Proc. of Intl. Conf. on High-Performance Parallel and Distributed Computing (HPDC)*, Monterey, CA, June 2007.

[28] Paolo Costa, Luca Mottola, Amy L. Murphy, and Gian Pietro Picco. TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In *Proc. of the Intl. Workshop on Middleware for Sensor Networks (MidSens)*, Melbourne, Australia, November 2006.

[29] Yun Huang, Shivajit Mohapatra, and Nalini Venkatasubramanian. An Energy-efficient Middleware for Supporting Multimedia Services in Mobile Grid Environments. In *Proc. of Information Technology: Coding and Computing (ITCC)*, Las Vegas, NV, April 2005.

[30] Nanyan Jiang, Cristina Schmidt, Vincent Matossian, and Manish Parashar. Enabling Applications in Sensor-based Pervasive Environments. In *Proc. of BaseNets in conjunction with Broadband Communications, Networks, and System (BroadNets)*, San Jose,CA, October 2004.

[31] Wen-Hwa Liao, Jang-Ping Sheu, and Yu-Chee Tseng. GRID: A Fully Location-Aware Routing Protocol for Mobile Ad Hoc Networks. *Telecommunication Systems*, 18(1-3):37–60, 2001.

[32] Preetam Ghosh, Kalyan Basu, and Sajal K. Das. A Game Theory-Based Pricing Strategy to Support Single/Multiclass Job Allocation Schemes for Bandwidth-Constrained Distributed Computing Systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(3):289–306, 2007.

[33] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working Models for Uncertain Data. In *Proc. of Intl. Conf. on Data Engineering (ICDE)*, Atlanta, GA, 2006.

[34] Yanlei Diao, Boduo Li, Anna Liu, Liping Peng, and Charles Sutton. Capturing Data Uncertainty in High-Volume Stream Processing. In *Proc. of Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, January 2009.

[35] Reynold Cheng, Jinchuan Chen, and Xike Xie. Cleaning Uncertain Data with Quality Guarantees. *Proc. of the VLDB Endow.*, 1(1):722–735, August 2008.

[36] Kyungyong Lee and Renato Figueiredo. MapReduce on Opportunistic Resources Leveraging Resource Availability. In *Proc. of IEEE Conf. on Cloud Computing Technology and Science (CloudCom)*, Taipei, Taiwan, December 2012.

[37] Luiz Fernando Bittencourt and Edmundo R.M. Madeira. Towards the Scheduling of Multiple Workflows on Computational Grids. *Journal of Grid Computing*, 8(3):419–441, 2010.

[38] Georgios L. Stavrinides and Helen D. Karatza. Scheduling Multiple Task Graphs in Heterogeneous Distributed Real-time Systems by Exploiting Schedule Holes with Bin Packing Techniques. *Simulation Modelling Practice and Theory*, 19(1):540 – 552, 2011.

[39] Ying-Lin Tsai, Kuo-Chan Huang, Hsi-Ya Chang, Jerry Ko, En Tzu Wang, and Ching-Hsien Hsu. Scheduling Multiple Scientific and Engineering Workflows through Task Clustering and Best-Fit Allocation. In *Proc. of IEEE World Congress on Services (SERVICES)*, Honolulu, HI, June 2012.

[40] Gyung-Leen Park, Behrooz Shirazi, and Jeff Marquis. DFRN: A New Approach for Duplication based Scheduling for Distributed Memory Multiprocessor Systems. In *Proc. of Intl. Parallel Processing Symp.*, Geneva, Switzerland, April 1997.

[41] Eun Kyung Lee, Hariharasudhan Viswanathan, and Dario Pompili. SILENCE: Distributed Adaptive Sampling for Sensor-based Autonomic Systems. In *Proc. of the ACM Intl. Conf. on Autonomic computing (ICAC)*, Karlsruhe, Germany, June 2011.

[42] Ivan Rodero, Francesc Guim, Julita Corbalan, Liana Fong, and S. Masoud Sadjadi. Grid Broker Selection Strategies Using Aggregated Resource Information. *Future Gener. Comput. Syst.*, 26(1):72–86, January 2010.

[43] Marcos Dias de Assunção, Rajkumar Buyya, and Srikumar Venugopal. Intergrid: a case for internetworking islands of grids. *Concurr. Comput. : Pract. Exper.*, 20(8):997–1024, June 2008.

[44] John D. C. Little. A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, May 1961.

[45] Ivan Rodero, Francesc Guim, Julita Corbalán, and Jesús Labarta. Design and Implementation of a General-Purpose API of Progress and Performance Indicators. In *Proc. of Intl. Conf. on Parallel Computing (ParCo)*, September 2007.

[46] K. Maruyama, S. K. Chang, and D. T. Tang. A General Packing Algorithm for Multidimensional Resource Requirements. *International Journal of Parallel Programming*, 6:131–149, 1977.

[47] Zhen Li and M. Parashar. Comet: A Scalable Coordination Space for Decentralized Distributed Environments. In *Proc. Intl. Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P)*, July 2005.

[48] Resource Provisioning in Mobile Grids.
http://nsfcac.rutgers.edu/cps/projects/uncertainty-aware-resource-provisioning-mobile-computing-grids-real-time-situ-data.

[49] Govind S. Mudholkar, Deo Kumar Srivastava, and Georgia D. Kollia. A Generalization of the Weibull Distribution with Application to the Analysis of Survival Data. *Journal of the American Statistical Association*, 91(436):1575–1583, December 1996.

[50] Hyunjoo Kim, Yaakoub el Khamra, Ivan Rodero, Shantenu Jha, and Manish Parashar. Autonomic Management of Application Workflows on Hybrid Computing Infrastructure. *Telecommunication Systems*, 19(2-3):75–89, February 2011.

[51] Jon G. Rokne. Interval Arithmetic and Interval Analysis: An Introduction. In Witold Pedrycz, editor, *Granular Computing*. Physica-Verlag GmbH, 2001.

[52] Feng-Tso Sun, Cynthia Kuo, Heng-Tze Cheng, Senaka Buthpitiya, Patricia Collins, and Martin L. Griss. Activity-Aware Mental Stress Detection Using Physiological Sensors. In *Proc. of Intl. Conf. on Mobile Computing, Application, and Services (MobiCASE)*, Santa Clara, CA, 2010.

[53] Alberto De Santos, Carmen Sanchez-Avila, Javier Guerra-Casanova, and Gonzalo Bailador-Del Pozo. Real-time Stress Detection by means of Physiological Signals. In Jucheng Yang and Norman Poh, editors, *Recent Application in Biometrics*. InTech, 2011.

[54] William J. Marshall and Stephen K. Bangert. *Clinical Biochemistry - Metabolic and Clinical Aspects*. Churchill Livingstone (Elsevier), USA, 2008.

[55] Jack Hayya, Donald Armstrong, and Nicolas Gressis. A Note on the Ratio of Two Normally Distributed Variables. *Management Science - Theory Series*, 21(11):1338–1341, 1975.

[56] Eduardo F. Nakamura, Antonio A. F. Loureiro, and Alejandro C. Frery. Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications. *ACM Computing Surveys*, 39(3), September 2007.

[57] Jochen Gorski, Kathrin Klamroth, and Stefan Ruzika. Generalized Multiple Objective Bottleneck Problems. *Operations Research Letters*, 40(4):276 – 281, 2012.

[58] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proc. of Symp. on Operating Systems Design and Implementation (OSDI)*, New Orleans, LA, February 1999.