A PROGRAMMING SYSTEM FOR SENSOR-DRIVEN SCIENTIFIC APPLICATIONS

BY NANYAN JIANG

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

Professor Manish Parashar

and approved by

New Brunswick, New Jersey

May, 2009

ABSTRACT OF THE DISSERTATION

A Programming System for Sensor-driven Scientific Applications

by Nanyan Jiang

Dissertation Director: Professor Manish Parashar

Technical advances are leading to a pervasive computational ecosystem that integrates computing infrastructures with embedded sensors and actuators, and giving rise to a new paradigm for monitoring, understanding, and managing natural and engineered systems – one that is information/data-driven.

This research investigates a programming system that can support such end-to-end sensor-based dynamic data-driven applications. Specifically, it enables these applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific and engineering processes and with other application components in an end-to-end experiment. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms. The former supports complex querying of the sensor system, while the latter enables development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilation, both via semantically meaningful abstractions. For the latter, we explore the temporal and spatial correlation of sensor measurements in the targeted application domains to tradeoff between the complexity of coordination among sensor clusters and the savings that result from having fewer sensors for in-network processing, while maintaining an acceptable error threshold. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of constraint resources and satisfy data requirement in the presence of dynamics and uncertainty.

The research presented in this thesis is evaluated using two application scenarios: (1) the management and optimization of an instrumented oil field and (2) the management and optimization of an instrumented data center. In the first scenario, the programming abstractions and systems software solutions enable end-to-end management processes for detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface. The overall goal is to ensure near optimal operation of the reservoir in terms of profitability, safety and/or environmental impact. In the second scenario, the autonomic instrumented data center management system addresses power consumption, heat generation and cooling requirements of the data center, which are critical concerns especially as the scales of these computing environments grow. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

Acknowledgements

This dissertation is the culmination of a lot of effort, and much of it would not have been possible if it were not for the guidance, support, help and friendship of many people.

First and foremost, I would like to express my utmost gratitude to my advisor Prof. Manish Parashar, for his willingness to challenge and encourage me. This journey would not have been so rewarding and memorable without his vision, resolve, enthusiasm, inspiration, wisdom, and guidance during the course of my studies. He always knew how to improve a paper or project with his innate understanding of any subject matter! I also thank him for the opportunity to present my research at various forums as well as demonstrating prototypes of research ideas on the testbeds. The experiences gained through these research activities with his guidance will undoubtedly be beneficial in my future career.

I must also thank the many professors and researchers that provided me advice and helped my research development during my studies at Rutgers. In particular, I am grateful to Prof. Yanyong Zhang for sharing her invaluable experiences and providing insightful suggestions with me. I would like to thank Prof. Dario Pompili for stimulating in-depth discussions with me. I would also like to thank Prof. Ivan Marsic and Prof. Hoang Pham for their valuable advice and feedback throughout different stages of my study.

I would like to thank many of my colleagues who helped me along the way. In particular, I have had the good fortune to work with other members of our research group, The Applied Software and System Laboratory (TASSL) and Center for Autonomic Computing (CAC). There were many occasions where discussions with them helped me make progress with my research. Special thanks to Andres Quiroz for many research discussions and kind help. I would like to thank the staff at the Center for Advanced Information Processing (CAIP), Department of Electrical and Computer Engineering, Ivan Seskar and ORBIT team/WINLAB for their assistance and support.

Outside of Rutgers, I would like to thank Guofei Jiang, Haifeng Chen and Kenji Yohsihira of NEC Laboratories America, Princeton, New Jersey. The summer that I spent with them really helped expand my horizons as a researcher. I thank Hector Klie from University of Texas at Austin, Texas, for his collaboration on the application of oil reservoir. I would also like to thank Eliot Feibush from the Princeton Plasma Physics Laboratory (PPPL) at Princeton, New Jersey, for his help and collaboration.

My parents who worked hard to give me the opportunities I had and who encouraged me to live up to my potential; I sincerely thank them for their perseverance and love. My sister's encouragement, enthusiasm and friendship have been invaluable to me.

Finally, I wish to express my love and gratitude to my husband for his infinite patience, support, and encouragement throughout my graduate school career. I could not have reached this milestone without them.

The research presented in this thesis was supported in part by National Science Foundation via grants numbers IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, and by Department of Energy via the grant number DE-FG02-06ER54857, and was conducted as part of the NSF Center for Autonomic Computing at Rutgers University.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Technical advances are rapidly leading to a revolution in the type and level of instrumentation of natural and engineered systems, and are resulting in a pervasive computation ecosystem that integrates computers, networks, data archives, instruments, observatories, experiments, and embedded sensors and actuators. This in turn is enabling a new paradigm for monitoring, understanding, and managing natural and engineered systems – one that is information/data-driven and that symbiotically and opportunistically combines computations, experiments, observations, and real-time information to model, manage, control, adapt, and optimize.

Several application domains, such as waste management [92], volcano monitoring [119], city-wide structural monitoring [63], habitat and environmental monitoring [81], and end-to-end soil monitoring system [109], are already experiencing this revolution in instrumentation, and can potentially allow new quantitative synthesis and hypothesis testing in near real time as data streams in from distributed instruments. However, (1) the data volume and rates, (2) the uncertainty in this data and the need to characterize and manage this uncertainty, and (3) the need to assimilate and transport required data (often from remote sites over low bandwidth wide area networks) in near real-time so that it can be effectively integrated with computational models and analysis systems, present significant challenges. As a result, data in most existing instrumented systems is used in a post-processing manner where data acquisition is a separate offline process.

1.2 Problem Description

In many current scientific and engineering applications, modeling to predict system behavior is largely done using static historical data. This approach makes it impossible for such models to accurately predict temporal and spatial variations in the real-world. With advances in sensor technology, it now becomes possible to feed in near real time measurements data from diverse, complex, distributed sensor networks, enabling more accurate modeling, prediction and control. This research investigates conceptual as well as systems software issues for enabling the integration of sensor systems with computational applications. Specifically, the research is driven by the management and control of subsurface geosystems, such as managing subsurface contaminants at the Ruby Gulch waste repository [92] and management and optimization of oil reservoirs [61]; and by the management and optimization of an instrumented data center [53]. Crosscutting requirements of these applications include:

Multi-scale, multi-resolution data access: Spatial and temporal variations in the phenomenon being understood and managed by the application requires data at multiple scales and resolutions at different locations of the monitored field. This requires online (near real-time) spatial and temporal interpolation of the data from sensors before it can be integrated with executing simulations.

Data quality and uncertainty estimation. Scientific investigation present strict requirements on data quality, uncertainty estimation and management. Further, the data assimilation and uncertainty estimation mechanisms should be near real-time and managed within the sensor system to be able to respond adequately and opportunistically to anomalous events.

Predictable temporal response to varying application characteristics. Ensuring robust sense-evaluate-actuate cycles of scientific/engineering processes requires low and predictable information (not just data) generation latencies. These requirements may differ from application to application and for different phases of the same application. For example, applications involving aerodynamic stabilization and neural control may require millisecond level response while geosystem management (such as the applications described above) may require responses in seconds, hours, days or weeks depending on the nature of the control task.

In addition to the above application requirements, there are a number of challenging system level requirements including scheduling and adaptive runtime management of in-network data processing, load balancing, quality of service management, resource management and computation/communication/energy tradeoffs. Furthermore, integrating sensors system with applications requires sufficiently high-level and easy-to-use programming abstractions and systems.

1.3 Approach

Addressing the issues outlined above requires a middleware infrastructure that can effectively extract and abstract desired information from the huge amounts of data present in sensor based systems, despite their scale, heterogeneity and dynamism. Further, it should automate some of the decision processes associated with the computation in the network. This research develops a programming system to support the development of in-network data processing mechanisms, and enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems using semantically meaningful abstractions. The programming systems enables sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it supports programming models and systems for developing in-network data processing mechanisms. The former should support complex querying of the sensor system, while the latter should enable development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilations, both via semantically meaningful abstractions. A key requirement here is being able to specify and enforce dynamic data requirements and quality of data and service constraints, as well as investigate tradeoffs between data quality, resource consumptions and performance.

1.4 Overview of the Programming System

The overall goal of this research is to develop sensor system middleware and programming support that will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing as they stream data from the physical environment to the computational world. Further, application should be able to interact with the sensor system to control sensing and data processing behaviors. The system software consists of three key components described below.

- **Programming System:** We will investigate a programming system that will support the development of in-network data processing mechanisms, and will enable scientific and engineering applications to discover, query, interact with, and control instrumented physical systems using semantically meaningful abstractions. The former will be based on content/location-based messaging, while the latter will provide associative querying services. The programming system will build on existing service and component-based programming models.

- **Data Processing Middleware:** We will investigate services for active in-network data processing that will generate appropriate data/information to drive novel algorithms for modeling, interpretation of phenomenon, and decision making. We will also investigate algorithms and mechanisms to enable the acquisition of this data/information with dynamic qualities and properties from streams of data from the physical environment, and address issues of data quality assurance, statistical synthesis and hypotheses testing, and in-network data assimilation. Finally, we will explore how applications can interact with sensors to control data acquisition.

- **Sensor System Management Services:** We will investigate middleware services for the application-driven dynamic management of sensor systems for physical instrumentation, including overlay management, runtime management of data processing including adaptations for computation/communication/power tradeoffs, dynamic load-balancing, system resource management, and sensor system adaptations for application driven data acquisition. The key idea will be to use application domain information to optimize adaptations and make appropriate tradeoffs.

1.5 Contributions

The key contribution of this work is that it provides a conceptual architecture model and sensor system middleware and programming support that will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing as they stream data from the physical environment to the computational world.

- The *GridMap/iZone* Programming System. This programming system includes abstractions and runtime mechanisms for integrating sensor systems with applications processes, as well as for in-network data processing such as aggregation, adaptive interpolation and assimilation. Specifically, the end-to-end abstractions provided by programming system is to enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems in a semantically meaningful way. For the latter, we explore the temporal and spatial correlation of sensor measurements in the targeted application domains to tradeoff between the complexity of coordination among sensor clusters. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of resources and satisfy data requirement in the presence of dynamics and uncertainty.

- A Decentralized Content-based Aggregation Services for Pervasive Grid Environments. This research presents the aggregation service that builds on the Meteor content-based middleware infrastructure [55] for content-based information discovery and decoupled interactions. Specifically, it extends Meteor to enable content-based aggregation queries to be flexibly specified using keywords, partial keywords and ranges. Further, it builds on a self-organizing overlay network and the Squid content-based routing infrastructure to construct aggregation tries so that query propagation routes can be used for back-propagating and aggregating matching data elements. The deployment and experimental evaluation of the aggregation service are conducted on a LAN, the wireless ORBIT testbed [2] at Rutgers University, and the PlanetLab wide-area testbed [93], which demonstrates the scalability, effectiveness of the system.

- Enabling End-to-end Sensor-driven Scientific and Engineering Applications. The research presented in this thesis is evaluated using two application scenarios: (1) the management and optimization of an instrumented oil field and (2) the management and optimization of an instrumented data center. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

1.5.1 Impacts

While there has been extensive research in sensor networks, the infrastructure required for *closing the loop* and *integrating sensor systems with computational applications*, where the sensor information can be used for near real-time analysis, understanding, decision making and actuation, is largely missing. We believe that such an infrastructure is critical and will lead to the development of a new generation of dynamic sensor driven ad hoc control systems that use sensor data to accurately predict the behavior of large scale natural and engineered systems, and proactively manage and control their operation. Such a dynamic data-driven and knowledge-based approach will increase productivity, reduce cost and improve safety.

1.6 Outline of the Dissertation

The rest of the thesis is organized as follows.

Chapter 2 gives an overview of existing programming support and abstractions for sensor networks. It discusses the limitations of these existing systems with respect to the requirements of sensor driven scientific and engineering applications. Furthermore, this chapter compares the key differences between *GridMap/iZone* with the related systems.

Chapter 3 presents the *GridMap/iZone* programming systems that enable sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific and engineering processes and with other application components in an end-to-end experiment. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms.

Chapter 4 investigates a programming system to support the development of in-network data processing mechanisms, which includes abstractions and runtime mechanisms for in-network data processing such as aggregation, adaptive interpolation and assimilation.

Chapter 5 describes a decentralized content-based aggregation service for pervasive grid environments. The deployment and experimental evaluation of the aggregation service are conducted on local area networks (LANs), the wireless ORBIT testbed [2] at Rutgers University, and the PlanetLab wide-area testbed [93], which demonstrate the scalability, effectiveness of the system.

The research presented in this thesis is evaluated using two application scenarios: (1) the management and optimization of an instrumented oil field and (2) the management and optimization of an instrumented data center. In Chapter 6, the programming abstractions and systems software solutions enable end-to-end management processes of an instrumented oil field consisting of detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface. The overall goal is to ensure near optimal operation of the reservoir in terms of profitability, safety and/or environmental impact.

In Chapter 7, the autonomic instrumented data center management system addresses power consumption, heat generation and cooling requirements of the data center, which are critical concerns especially as the scales of these computing environments grow. Sensor networks monitor temperature, humidity, and airflow in real time, and provide non-intrusive and fine-grained data collection, and enable real-time processing. These sensors are integrated with computational processes and job schedulers to take phenomenon, such as heat distribution and air flows into consideration, and to optimize data center performance in terms of energy consumption and throughput. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

Chapter 8 concludes the thesis and presents future research directions.

Chapter 2

Background and Related Work

This chapter investigates the current programming systems for sensor networks and pervasive grid systems. An overview of existing programming abstractions and support for sensor networks is presented in Section 2.1. We discuss these existing programming systems with respect to the requirements of sensor driven scientific and engineering applications. In-network processing mechanisms are discussed and compared with our work. Furthermore, we compare the key differences between *GridMap/iZone* programming system with the related systems. In Section 2.2, content-based middleware is presented and compared with the Meteor middleware infrastructure and the aggregation service. Finally, Section 2.3 introduces the sensor system management technologies that enable application-driven dynamic management of sensor systems for physical instrumentation.

2.1 Programming Abstractions and Support

2.1.1 Programming Support and Abstractions for Sensor Networks

There has been a significant body of research on programming support for sensor networks, for both, interfacing with them and developing application that run on them. Early approaches were relatively low-level where application mechanisms were hard coded into the devices [6], or were based on specialized device operating systems (e.g., TinyOS [70], SOS [43], Contiki [23]), languages (e.g., NesC [35]) or virtual machines (e.g., Mate [67], Deluge [48], Trickle [71], Tofu [68]). Other efforts such as MiLan [46] and Impala [76] specifically addressed several challenges of wireless sensor networks, focusing on the long-lived nature and resource-constrained, dynamic and heterogeneous environment of sensor applications. TOSSIM [69] and Emstar [37] provide simulation environments for developing and deploying wireless sensor networks. Trio [25] provides tools for deploying sensor networks. TinyGals [19] is an event-driven programming model for embedded system through synchronous method calls to form modules and asynchronous message passing between modules to separate control of flows.

Recent programming systems can be classified based on the abstractions they provide. *Messaging-oriented* approaches primarily provide communication abstractions that can be used to build sensor applications. This category includes system providing low level abstraction (e.g., Directed diffusion [49] and Dfuse [64]) as well as higher level abstraction (e.g., publish-subscribe and content-based rendezvous [55]). *Mobile agent-based* approaches allow in-network reprogramming through mobile agent migration. Systems in this category include Agilla [28] and Agimone [42]. The latter targets the integration of sensor and IP networks across heterogeneous devices.

Several systems [73,78,117,121] provide abstractions for specifying local behaviors of sensors. EnviroSuite [78], state programming [73] and CLB [56] are based on emergence, where the programmer specifies local sensor behaviors the global behaviors emerge. For example, EnviroSuite is targeted at tracking applications. Abstract region [117] and Hood [121] support a neighborhood-based programming models in a sensor network. These systems are particularly relevant to the requirements addressed in this research.

In the alternate *macroprogramming* approach, global behaviors are specified and the programming system generates the local behaviors and interactions necessary, such as Regiment [88], TML [87], Semantic Streams [120], Kairos [40] and ATaG [7]. For example, Semantic Streams provides a logic-based language for composing distributed data-processing services. *Database-oriented* approaches provide abstractions that view the sensor network as a virtual database system and provide SQL-like interfaces for querying the networks, including Cougar [124] and TinyDB [80]. The related *data streaming* approach supported by TelegraphCQ [17] views data as information data streams, and applications monitor and react to them as they pass through the network.

The programming systems discussed above have to be extended to support end-to-end sensor-driven applications and the interactions between computational models and the sensor system, and address requirements discussed in Section 1.2. Ideally, a scientist should only have to specify application data requirements using high-level abstractions, and the system should transform these requirements into appropriate operations, interactions and coordination within the sensor system. The *GridMap/iZone* is such programming system to provide semantically meaningful abstractions and runtime mechanisms for integrating sensor systems with computational models for scientific processes, as well as for in-network data processing such as aggregation, adaptive interpolation and assimilations.

2.1.2 Resource Aware Programming System

Resource aware programming [85] is an important aspect of system software, especially for resource constraint sensor networks. The *GridMap/iZone* programming system is closely related to a range of primitives for resource management in sensor networks, and resource adaptivity for mobile and pervasive computing systems.

Many approaches have relied on hardware support for energy estimation, such as Eon [105], iCount [24] and Triage [9]. For example, Triage [9] allows application logic to be partitioned across coupled hardware platforms based on energy availability. The driver architectures ICEM [62] and configurational software kit SNACK [38] automate some aspects of energy savings by coordinating access to hardware resources across multiple components and concurrent tasks. Adaptive duty-cycling algorithms [59,116] tune the duty cycle of a sensor network application according to energy availability. The *iZone* software is orthogonal to these approaches and manage the energy for a set of sensors and take the quality of estimation into consideration.

The *GridMap/iZone* system is related to several systems by taking resource-aware application design into consideration. TinyDBs queries [80] target a given lifetime by (statically) setting the query duty cycle. Levels [65], Eon [105] and Pixie [77] provides programming models for adapting energy availability. Pixie is an operating system and a *dataflow* programming model that permits high-level, reusable resource management policies via resource tickets and brokers [77]. Eon provides a dataflow model similar to Pixie and automatically tunes timer rates and dataflow paths based on energy availability. Levels allows application components to define multiple fidelity levels, which are configured in response to energy availability. Contiki [23] uses a model to track hardware states in software.

The *GridMap/iZone* system focuses *coordinated* resource management of sensor nodes across the network and quality of service specified by the applications by providing extensive runtime adaptation. Specifically, the *GridMap/iZone* programming system allows such policies to be expressed through a unified programming model that can permit domain scientists and other non-expert users to build adaptive, efficient, and self-tuning sensor networks by specifying and implementing resource management with quality of service as a core aspect of the programming abstraction.

The sensor selection schemes are also closely related to our work. The sensor selection approach described in [22] uses approximation algorithms to select near-optimal subsets of k sensors that minimize the worst-case prediction error. Entropy-based approaches [126] are used for the sensor selection problems of target tracking and localization applications. The goal of our proposed algorithms is to find a *"best" collection* of subsets, *all* of which satisfy the error tolerance rate (and minimize aggregated errors), while saving and balancing the energy consumptions among sensors in the long run.

2.1.3 In-network Processing and Virtual Sensors

Data aggregation is an essential functionality in sensor networks, and has been addressed by a number of research efforts [79, 124]. Optimizations techniques such as aggregation trees are used to resolve queries efficiently. Homogeneous aggregation operations are supported. The approach presented in this research supports used-defined functions using in-network coordination and optimization mechanisms.

Other related efforts include spatial interpolation and aggregation [30, 33, 103] and redundant sensor sampling [72]. The focus of this work is different in that it develops programming abstractions to facilitate programming the in-network algorithms (e.g. varied interpolation algorithms), as well as runtime mechanisms to investigate trade offs between dynamic coordination costs and data quality. The Lance [118] is used to optimize collecting raw sensor data from large networks under varying energy and bandwidth constraints. This system is complementary to *GridMap/iZone* system, although our focus is on trading energy consumptions and quality requirements in a long run.

There are some recent research efforts [3, 14, 58, 86, 91] that use the concept of virtual sensors to support sensor applications. VNLayer [14] provides abstraction layers that mask uncertainty of underlying sensor networks through consistency management. Virtual sensor [58] provides a virtual sensor model and application APIs to support heterogeneous aggregation and hierarchical specifications. However, the underlying concepts and implementation of the system described in this thesis is quite different from these approaches in that it uses virtualization to address the mismatch between the instrumentation of the physical domain and its discretization in the computational model, rather than to create, for example, a virtual sensor for a derived data type.

2.1.4 End-to-end Sensor-based Applications and Systems

The soil ecology monitoring system [109] is an end-to-end data collection prototype that uses wireless sensor systems as the first component of an end-to-end system. The ring buffer network bus (RBNB) DataTurbine is a streaming middleware system [113], of which the key components are the ring buffers and network bus objects for managing, archiving and accessing data from local and remote produces (e.g., instruments, users). Que [20] provides a scripting-based exploration environment to support simulation and emulation of multi-platform tiered systems including sensor network for real time data acquisition.

The *GridMap/iZone* system provides programming abstractions and runtime support to integrate the different components of sensor-driven applications. The system provides end-to-end and in-network support for integrating sensor systems with computational process by essentially virtualizing the sensor field to match its representation used by the computational model.

2.2 Content-based Middleware Infrastructure

The growing ubiquity of sophisticated sensor/actuator devices with embedded computing and communications capabilities, and the emergence of Grids are resulting in a pervasive information infrastructure that combines computational, storage and information resources [26, 36]. This pervasive infrastructure is, in turn, enabling new generations of information/data-driven applications that are based on seamless access, aggregation and integration.

Content-based Interaction: Content-based decoupled interactions have been addressed by publish-subscribe-notify (PSN) models [27]. PSN based systems include Sienna [16] and Gryphon [39]. The associative rendezvous model differs from PSN systems in that individual interests (subscriptions) are not used for routing and do not have to be synchronized - they can be locally modified at a rendezvous node at anytime.

The i3 [107] provides a similar rendezvous-based abstraction and has influenced this work. However, an i3 identifier is opaque and must be globally known. Associative rendezvous uses semantic identifiers that are more expressive and only require the existence of agreement upon information spaces (ontologies). Besides, its dynamic binding semantics enables profiles to be added, deleted or changed on-the-fly.

The associative broadcast [12] paradigm has also influenced this effort. The key difference between this model and associative rendezvous is that the binding of profiles takes place at intermediate nodes instead of the broadcast medium. As a result, associative broadcast only supports transient interactions. Further, its scalability over wide areas is a concern.

The rendezvous-based communication is conceptually similar to tuple space research in distributed systems [50, 89, 123]. A tuple space is a shared space that can be associatively accessed by all nodes in the system. While tuple space is a powerful model for interactions and coordination, efficient and large-scale implementations of pure tuple space based systems is a challenge. Associative rendezvous maintains the conceptual expressiveness of tuple spaces while providing an implementation model that is scalable.

Unlike other rendezvous-based models [34], associative rendezvous enables programmable reactive behaviors at rendezvous points using the action field within a message. In addition, associative rendezvous is able to realize a variety of basic communication services without the need for mobile code [112], or any heavy duty protocols. Further, interactions in the associative rendezvous model are symmetric allowing participants to simultaneously be information producers and consumers.

Narada Brokering [29] is a distributed middleware framework that supports peer-to-peer systems and content-based publish/subscribe interactions. It manages a network of brokers through which end systems can interact, providing scalability, location independence, and efficient content-based querying and routing. However, Narada brokers are organized in a hierarchical structure, which is maintained through tighter coupling and control mechanisms, focusing on persistence and reliable message delivery. In contrast, Meteor is meant to support more dynamic and opportunistic interactions in a peer-to-peer network.

Content-based publish/subscribe over DHT is a topic for which there is much current work. DHT functionality is usually built using some sort of a structured overlay network, the most popular of which are Chord [108], used here, Pastry [98], and CAN [96], because they provide scalability, search guarantees and bounds on messaging within the network, as well as some degree of self-management and fault tolerance with respect to the addition/removal of nodes. With this foundation, designing content-based publish/subscribe systems requires an efficient mapping between content descriptors and nodes in the overlay network, as well as efficient techniques for routing and matching based on these content descriptors, which can contain wildcards and ranges for complex queries. The work in [4, 8, 41, 110] addresses these issues to some extent. Meteor and Squid differ from these approaches mainly in the locality-preserving mapping used.

The Meteor framework has recently been used to support a Web Services based notification broker service for content-based subscription management and notification dissemination targeting highly dynamic pervasive Grid environments that adopt the Web service notification (WSN) standards [94]. This service makes use of Meteor's AR messaging and reactive behaviors to provide a distributed and decentralized implementation of the operations defined by the WSN interfaces.

Programming Models for Sensor-based Pervasive Systems The Cornell Cougar [124], TinyDB [80] and Tiny Aggregation (TAG) [79] systems provide high level programming abstractions that view the sensor network as a distributed database, and provides SQL-like interfaces for querying the networks. Optimizations techniques such as aggregation trees are used to resolve queries efficiently. Only single tasking with homogeneous aggregation operations is supported. The approach presented in this research defines the aggregation operations as programmable reactive behaviors and can associate different aggregation operators with different data scopes and properties.

Other related work include TelegraphCQ [17], which uses window-based query semantics for continuous queries; it uses an efficient filtering mechanism corresponding to the desired end-to-end application behavior. The original design is primarily a single node system. Our system, on the other hand, distributes queries in the network. Songs [75] provides service-oriented architecture to convert declarative user queries into a service composition graph, for sensor-based pervasive system.

Aggregation in Peer-to-Peer (P2P) Environments: Aggregation in large P2P distributed systems is often based on a hierarchical architecture. The Astrolabe [114] project at Cornell is designed to provide a DNS-like distributed management service to allow nodes to aggregate information by dividing the network into (non-overlapping) zones arranged hierarchically. Similarly, the effort presented in [11] formulates the Node Aggregation problem in P2P systems and presents a number of approaches to address the problem, including approaches based on spanning tree induction and using redundant topologies. While these approaches are similar to the one presented in this research, the key difference is the support for content-based aggregation query formulations and the aggregation guarantees provided. Cone [13] augments a DHT with a trie to support heap functions, which provide an aggregation operator at the root of every subtree. For different aggregate operators and applications, different independent tries and/or overlay structures can be formed. In this research, a generic prefix trie-based aggregation protocol is provided, which supports content-based in-network aggregations using different application-specific operators or functions using the same trie structure. Sharing the same trie structure among multiple queries and applications amortized maintenance costs.

PHT [18] is based on a trie data structure, which requires explicit periodic maintenance of the underlying trie structure. Our approach presented in this research also usess a trie-based approach; however, unlike PHT, our approach maintains no persistent state about the aggregate trie in the overlay.

2.3 Sensor System Management Technologies

In this section, we will investigate the related work of middleware services for the application-driven dynamic management of sensor systems for physical instrumentation, including overlay management, runtime management of data processing including adaptations for computation/communication/power tradeoffs, dynamic load-balancing, system resource management, and sensor system adaptations for application driven data acquisition. The key idea is to use application domain information to optimize adaptations and make appropriate tradeoffs.

The structure of the logical topology that connects and organizes sensors can have significant impact on the efficiency of communication and computations, especially in the presence of large data volumes and constrained resources.

Overlay networks have been addressed extensively in wired networks, and several unstructured (e.g., Gnutella Network, Freenet [21]) and structured solutions (e.g., (SINA [106]), DHT-based (CAN [96], Chord [108], SquidTON [102], Willow [115]), such as distributed data structure based (SkipNet [44], Cone [13]) and attribute based (e.g. Direct Diffusion [49])), have been proposed with tradeoff between management overheads and communication guarantees. However, these effort are not directly applicable to wireless sensor networks, primarily due to resource (connectivity, bandwidth, energy) constraints.

The communication costs as well as error rates in wireless sensors networks depends on the physical distances between sensors, and as a result, an ideal sensor topology must be aware of the physical topology of the sensor network while at the same time providing the expressiveness of a structured overlay. Cluster-based topologies have been widely used in sensor networks, for the design and implementation of network protocols and collaborative signal processing applications for WSNs (e.g., [104,125]), primarily due to their simplicity, flexibility, and robustness. For example, the Low-Energy Adaptive Clustering Hierarchy (LEACH) [45] randomly divides the sensor network into several clusters, each cluster being managed by a cluster head. A more recent approach, which is based on ideas from structured overlays, is the virtual routing ring (VRR) [15]. VRR is inspired by Tapestry ring overlay, and takes advantage of the physical paths in addition to logical paths for routing. Setup and maintenance costs still have to be addressed in this work. Multi-scale overlay [90] designs a self-organizing hierarchical overlay that scales to a large number of sensors and enables multi-resolution collaboration. Another related effort is multi-resolution storage [32], which addresses in-network storage and distributed search in sensor networks.

In this project, we build on these existing solutions and sensors network topologies and address the challenges of in-network processing, storage and query resolution. A key issue is the maintenance of locality, both physical locality in the instrumented region, and logical locality in the application information space. This means that data is stored and processed at or close to the location where it is produced. Further, application queries should also, as far as possible, map to localized regions of the sensor network. As mentioned above, the developed solutions will try to use domain knowledge if available. For example, in most scientific simulations are based on a discretization of the physical domain and communication and computations are based on and demonstrate locality within this discretized domain. By indexing the sensor data so as to take this locality into account, e.g., using SFC's, the routing and querying behaviors can be optimized [57], which is presented with Meteor infrastructure and the aggregation services.

Another key issue is the adaptive runtime management of the querying and data processing operations so as to meet application performance requirements. The selection and scheduling of these operations should be system-sensitive, i.e., aware of the state and capabilities of the sensor nodes and communication system. Several existing sensor infrastructures (e.g., [31, 46, 76]) are designed to constrain the active components for achieving performance/resource tradeoffs. We extend this approach by using information about the domain, for example, many measured properties are relatively smoothly distributed and do not change abruptly. As a result, sensors in homogeneous regions (with respect to such properties) can be selectively adaptively activated to better manage energy based on desired performance/energy/communication tradeoffs.

Chapter 3

The *GridMap/iZone* Programming System

Technical advances are leading to a pervasive computational infrastructure that integrates computational processes with embedded sensors and actuators, and giving rise to a new paradigm for monitoring, understanding, and managing natural and engineered systems - one that is information/data-driven. However, developing and deploying these applications remains a challenge, primarily due to the lack of programming and runtime support. This chapter addresses these challenges and presents a programming system for end-to-end sensor/actuator-based scientific and engineering applications. The programming system provides semantically meaningful abstractions and runtime mechanisms for integrating sensor systems with computational models for scientific processes, and for in-network data processing such as aggregation, adaptive interpolation and assimilations. The overall architecture of the programming system and the design of its key components, as well as its prototype implementation are described.

3.1 Introduction

Sensor system middleware and programming support will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing as they stream data from the physical environment to the computational world [52]. Further, application should be able to interact with the sensor system to control sensing and data processing behaviors. The programming systems enables sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it supports programming models and systems for developing in-network data processing mechanisms. The former supports complex querying of the sensor system, while the latter enables development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilations, both via semantically meaningful abstractions. The research is driven by the management and control of subsurface geosystems, such as managing subsurface contaminants at the Ruby Gulch waste repository [92] and management and optimization of oil reservoirs [61], and by the management and optimization of an instrumented data center [53]. Crosscutting requirements of these applications include multi-scale, multi-resolution data access, data quality and uncertainty estimation, and predictable temporal response to varying application characteristics.

The focus of this chapter is on the end-to-end abstractions provided by programming system, and on how they can be used to enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems in a semantically meaningful way. Specifically, this chapter describes the design and operation of the *GridMap/iZone* abstractions.

The rest of the chapter is organized as follows. Section 3.2 gives and overview of the system. Section 3.3 introduces the *GridMap/iZone* approach for enabling end-to-end scientific and engineering applications. Section 3.4 presents the programming system architecture. Section 3.5 describes the design and operation of the *GridMap/iZone* programming abstractions. Section 3.6 presents an overview of the implementations. Section 3.7 summarizes the chapter.

3.2 A Programming System for Sensor-based, Dynamic Data-Driven Applications

A conceptual overview of the overall approach is illustrated in Figure 3.1. The goal of the programming system being developed as part of this research is to provide abstractions and mechanisms to seamlessly access and integrate remote and distributed sensor data into computational models and support scalable in-network data processing. The underlying approach is to virtualize the physical sensor grid to match the representation of the physical domain used by the models, and dynamically discover and access sensor data independent of any change to the sensor network itself. The sensor network periodically estimates data at the grid points specified by the application using available physical data values. The estimation mechanisms are

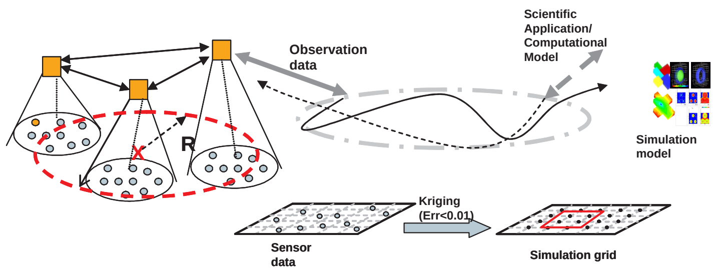


Figure 3.1: Overview of the programming system for sensor-driven applications.

specified by the applications and are implemented within the sensors network in a decentralized and scalable manner.

The abstractions provided enable applications to query the sensor system for data/information using flexible content descriptors that are semantically meaningful. For example, in a simulation defined on a regular catesian grid, a computational model looking for data in a particular region may specify this query using the coordinates that define the region in the computational grid and details about the data required (i.e., type, resolution, etc.).

Related Work in Programming Systems for Sensor Networks There has been a significant body of research focused on programming support for sensor networks, for both, interfacing with them and developing applications that run on them. Recent efforts are broadly classified below. Several systems [73, 78, 117, 121] provide abstractions for specifying the local behaviors of sensors. EnviroSuite [78], state programming [73], and CLB [56] are based on emergence, where the programmer specifies local sensor behaviors and global behaviors emerge from local behaviors and interactions. Abstract region [117] and Hood [121] provide neighborhood-based programming models for sensor networks.

In the *macroprogramming* approach, global behaviors are specified and the programming system generates the local behaviors and necessary interactions, e.g., Kairos [40] and ATaG [7]. *Database-oriented* approaches provide abstractions that view the sensor network as a virtual database system and provide SQL-like interfaces for querying the networks, e.g., Cougar [124] and TinyDB [80]. The related *data streaming* approach, supported by TelegraphCQ [17], views data as information data streams, and applications monitor and react to them as they pass through the network.

3.3 The *GridMap/iZone* Approach

The programming systems discussed above have to be extended to support end-to-end sensor-driven applications and the interactions between computational models and the sensor system. They must address the mismatch between the locations of the sensors and the data requirements of the computational models, the dynamic nature of application requirements and of the sensor system, data uncertainty and application constraints on quality of data and service. Furthermore, the system software should be able to actively support intelligent processing, such as adaptive interpolations, assimilations, which are needed by the models. Ideally, a scientist should only have to specify application data requirements using high-level abstractions, and the system should transform these requirements into appropriate operations, interactions and coordination within the sensor system to transform the sensed data to match these requirements.

While systems such as Kiaros [40] and TelegraphCQ [17] address some aspects of these requirements, the key difference is in the type of querying and processing supported. A key aspect of the *GridMap* approach presented in this research, is the indexing of data in the sensor system in a domain and locality aware manner using meaningful content descriptors. The descriptors are derived from the application domain and are used to define a semantically specified information space [100], which is the basis for all interactions with and within the sensor system. Note that many such information spaces can co-exist corresponding to different applications. This enables applications to query the sensor system for data/information using flexible content descriptors that are meaningful. For example, an oil reservoir simulation looking for well data in a particular region may specify this query using the coordinates that define the region, the type of well (production or injection) and details of the data required (i.e., type, resolution, etc.). Note that the content descriptors may also include ranges and wildcards. This abstraction can also specify data processing operations such as aggregations and interpolations and these operations can also be localized to specific regions in the sensor field.

There also exist recent research efforts that are similar to the presented approach in their use of virtual sensors [3, 14, 58, 86, 91] to support sensor-based applications. However, the underlying concepts and implementation of the system described in this research is quite different from these approaches in that it uses virtualization to address the mismatch between the instrumentation of the physical domain and its discretization in the computational model, rather than to create, for example, a virtual sensor for a derived data type. Other related efforts include spatial interpolation and aggregation [30, 33, 103], and redundant sensor sampling [72]. The focus of this work is different in that it addresses programming abstractions to facilitate implementations of in-network data estimation algorithms (e.g. various interpolation algorithms), as well as runtime mechanisms to investigate tradeoffs between dynamic coordination costs and data quality.

3.4 System Architecture

Key requirements for programming systems for enabling end-to-end sensor-driven applications and supporting the interactions between computational models and the sensor system include addressing the mismatch between the locations of the sensors and the data requirements of the computational models, the dynamic nature of application requirements and of the sensor system, data uncertainty and application constraints on quality of data and service. Furthermore, the system software should be able to actively support intelligent processing, such as adaptive interpolations, assimilations, which are needed by the models. Ideally, a scientist should only have to specify application data requirements using high-level abstractions, and the system should transform these requirements into appropriate operations, interactions and coordination within the sensor system to transform the sensed data to match these requirements.

The overall goal of the programming system developed as part of this research is to provide abstractions and mechanisms to address the above requirements and enable computational models to seamlessly access and integrate remote sensor data.

A schematic overview of the overall programming system architecture is presented in Figure 3.2. It consists of a two-level programming abstraction, the end-to-end *GridMap* programming abstraction that enables computational applications to access and integrate sensor data into their models, and the in-network *iZone* programming abstraction to enable the development of scalable in-network data processing mechanisms.

The underlying middleware provides an in-network data processing engine, which supports

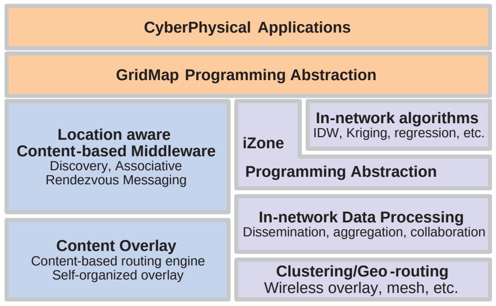

CyberPhysical Applications
GridMap Programming Abstraction
Location aware Content-based Middleware
Discovery, Associative Rendezvous Messaging
iZone
In-network algorithms
IDW, Kriging, regression, etc.
Programming Abstraction
In-network Data Processing
Dissemination, aggregation, collaboration
Content Overlay
Content-based routing engine
Self-organized overlay
Clustering/Geo-routing
Wireless overlay, mesh, etc.

Figure 3.2: A schematic overview of the programming system.

efficient data dissemination, aggregation and collaboration in dynamics, resource-constraint heterogeneous sensor networks, and a location-aware content-based middleware, for wide-area decentralized content-based discovery, associative rendezvous messaging and aggregation services for pervasive environments

3.5 The *GridMap* & *iZone* Programming Abstractions

Scientific applications often require measurements at pre-defined grid points, which are often different from the locations of the raw data provided directly by the sensor network. As a result, the sensor-driven scientific and engineering applications require a virtual layer, where the logical representation of the state of the environment provided to the applications may be different from the physical representation of raw measurements from sensor network. The abstractions described in this section enable applications to specify such a virtual layer and the models (e.g. regression models, interpolation functions, etc.) that should be used to estimate data on the virtual layer from sensor readings, as well to develop in-network implementations of the data estimation mechanisms.

The *GridMap* Abstraction

The *GridMap* abstraction consists of two operators. The first operator allows the application to construct a virtual grid (a *GridMap*), corresponding to the computational grid used by the computational models, on the instrumented domain. Once this virtual grid has been overlayed on the sensor system, the application can use the second operator to query data corresponding to a region of interest on this virtual grid. The interface of this second operator includes a

Operator	Semantics
query	Send query of *GridMap*
notify	Notify the status of *GridMap*
retrieve	Retrieve values of *GridMap*
init	Initialize grid points of *GridMap*
delete	Delete *GridMap*
refine	Refine the *GridMap* without reconstruction
coarsen	Coarsen the *GridMap* without reconstruction

Table 3.1: The *GridMap* operators.

specification of the method (e.g., interpolator) that should be used to estimate data at a grid point in the region of interest using physical data from sensors that are in the neighborhood of the point. The operator also includes parameters such as the size of neighborhood that should be used in the estimation, and what are the constraints on the accuracy and cost of the estimation.

The *GridMap* operators include end-to-end query operations, i.e., *query*, *notify*, *retrieve* as well as operators to construct, modify and delete the *GridMap* , i.e., *init*, *delete*, *refine* and *coarsen*. These operators are summarized in Table 3.1.

The parameters of the *query* operator include a specification of the region of interest within the *GridMap* and the interpolation function that should be used to compute values at the grid points of interest form the sensor values. The execution of this operator results in a query message being routed to relevant nodes (i.e., cluster heads) in the sensor network. The query specification is then matched against existing profiles, and if required, appropriate in-network operators are invoked. The *notify* operator is used to register notification requests, for example, and application may be interested in be notified if the maximum sensor reading in a region of the *GridMap* exceeds a certain threshold. The application can retrieve previously queried values using the *retrieve* operator.

The *init* operator is used to initialize the grid points associated with *GridMap*. The actual initialization steps are implementation specific. However, the initialization should return whether each grid point of the *GridMap* would be able to be constructed successfully. The success of *init* is an aggregation of successful construction of each grid point. If partial *GridMap* cannot be constructed successfully due to the unsuccessful construction of individual grid point, either a best-effort construction is rendered by the application or a failure flag is reported to the application. The *refine* operator modifies an existing *GridMap* by adding more grid points to increase the resolution of the *GridMap*. The *coarsen* operator modifies an existing *GridMap* by suspending some of the virtual grid points to effectively reduce the resolution of the *GridMap*. Note that, both *refine* and *coarsen* operators do not re-construct of the entire *GridMap* which makes them more efficient in dynamically changing physical environments.

Operator	Semantics
discover	Discover sensors of *iZone* specification
expand	Add more sensors to expand *iZone* coverage
shrink	Remove sensors to shrink *iZone* coverage
get	Return data from sensor(s) in the *iZone*
put	Send data to sensor(s) in the *iZone*
aggregate	Aggregate sensor data with MAX, MIN, WEIGHTED_SUM, etc.

Table 3.2: The *iZone* operators

The *iZone* Abstraction

The *iZone* abstraction in turn, enables the implementation of the estimation functions. Note that, user-defined functions can be implemented using *iZone* operators, which can then be applied in a straightforward fashion as a function operator on the actual running environment with *GridMap* operators as shown in Figure 3.3.

The *iZone* itself is a representation of the neighborhood that is used to compute a grid point, and may be specified using a range of coordinates, a function, etc. The *iZone* abstraction also provides operators, such as *discover*, *expand* and *shrink* for obtaining sensors corresponding to the region of interest as well as for defining in-network processing operators, such as *get*, *put* and *aggregate*, to compute a desired grid point from sensor values from this region as summarized in Table 3.2. The *discover* operator initially identifies and discovers sensors to manage the estimation. The operators such as *expand* and *shrink* identify, discover and update the participation of sensors at runtime. The *expand* operator increases the *iZone* by discovering more sensors without flooding the whole *iZone*. The *shrink* operator reduces the *iZone* by remove sensors that does not match the updated specification of *iZone* without re-discovering the whole *iZone*. The *get(p)* operator returns data matching profile *p* from the *iZone*. For example, the profile *p* can be specified as measurements whose locations are in the region of northeast from given grid point in the *iZone*. The *put(p)* operator sends the data to destination(s) matching profile *p* in the *iZone*. The *aggregate* operator collect relevant data and aggregate the partial results in the path to the destination.

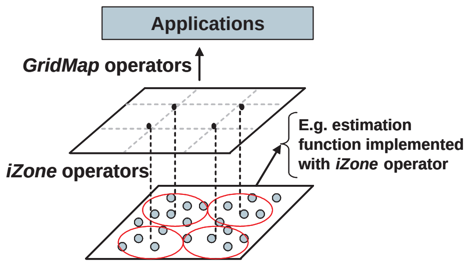


Figure 3.3: Use of *GridMap/iZone* operators

The *GridMap* and *iZone* abstractions thus abstract away the details of the underlying measurement infrastructure and hide the irregularities in the data by using a virtualization of the sensor field and estimation methods, to present a consistent representation, over time and space, to applications using the data. Once an *iZone* is defined, computing the data value at a grid point consists of (1) identifying a coordinator, which could be the sensor node that is closest to the grid point and has the required capabilities, (2) discovering the sensors in the *iZone* that will be used in the estimation, (3) planning the in-network estimation strategy based on desired cost/accuracy/energy tradeoffs, and (4) executing the estimation and returned the computed data value at the grid point. These are described in more details in the next chapter.

3.6 Implementation Overview

The current prototype implementation of *GridMap/iZone* programming system consists of two key parts. The sensor network component is implemented using the 802.11 protocol and standard location based clustering to construct a two level self-organizing overlay of sensor. This component implements the mechanisms for sensor discovery, query dissemination, data gathering and aggregation and in-network data processing. It has been prototyped using sensors emulated on the Orbit wireless testbed [2]. The wide-area component is built using Java and on top of the JXTA peer-to-peer substrate [1] and deployed on the Rutgers campus Grid. This component integrates computations processes (i.e. simulations), data archives and user subsystem to the sensor system through gateway nodes. Queries issued by the computational process are routed to the appropriate sensor nodes (and aggregated and interpolated data values routed) via the gateway and cluster heads.

The operation of the sensor network component of the *GridMap/iZone* programming system consists of bootstrap and running phases. The bootstrap phase is used to setup the sensor overlay. During this phase, sensor nodes form clusters and exchange information to setup routines tables, etc. The running phase consists of stabilization and user modes. In the stabilization mode, sensor nodes manage the structure of the clusters, and respond to periodic queries from other clusters to ensure that the routing tables are up-to-date, and to verify that sensor nodes have not failed or left the system. In the user mode, nodes respond to application requests. Further details of the operations of the two components as well as their evaluations can be found with subsequent chapters.

3.7 Summary

This chapter presented a programming system, that enables sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms. The overall architecture of the programming system and the design of its key components, as well as its prototype implementation were also described. In the next chapter, we investigate in-network data processing mechanisms with dynamic data requirements in resource constrained heterogeneous sensor networks.

Chapter 4

In-network Data Estimation for Sensor-driven Scientific Applications

Sensor networks employed by scientific applications often need to support localized collaboration of sensor nodes to perform in-network data processing. This includes new quantitative synthesis and hypothesis testing in near real time, as data streaming from distributed instruments, to transform raw data into high level domain-dependent information. This chapter investigates in-network data processing mechanisms with dynamic data requirements in resource constrained heterogeneous sensor networks. Particularly, we explore how the temporal and spatial correlation of sensor measurements can be used to trade off between the complexity of coordination among sensor clusters and the savings that result from having fewer sensors involved in in-network processing, while maintaining an acceptable error threshold. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of resources and satisfy data requirement in the presence of dynamics and uncertainty.

4.1 Introduction

The sensing technologies are rapidly leading to a revolution in the type and level of instrumentation of natural and engineered systems, and is resulting in pervasive computational ecosystems that integrate computational systems with these physical systems through sensors and actuators. This instrumentation can also potentially support new paradigms for scientific investigations by enabling new levels of monitoring, understanding and near real-time control of these systems. However, enabling sensor-based dynamic data-driven applications presents several challenges, primarily due to the data volume and rates, the uncertainty in this data, and the need to characterize and manage this uncertainty. Furthermore, the required data needs to be assimilated and transported (often from remote sites over low bandwidth wide area networks) in near real-time so that it can be effectively integrated with computational models and analysis systems. As a result, data in most existing instrumented systems is used in a post-processing manner, where data acquisition is a separate offline process.

In this chapter, we develop sensor system middleware and programming support that will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing, as they stream data from the physical environment to the computational world [52]. This chapter specifically investigates abstractions and mechanisms for in-network data processing that can effectively satisfy dynamic data requirements and quality of data and service constraints, as well as investigate tradeoffs between data quality, resource consumptions and performance. We first present the *iZone* programming abstractions for implementing in-network data estimation mechanisms. We then explore optimizations that can use the spatial and temporal correlation in sensor measurements to reduce estimation costs and handle sensor dynamics, while bounding estimation errors. For example, an appropriate subset of sensors might be sufficient to satisfy the desired error bounds, while reducing the energy consumed. The optimized in-network data estimation mechanisms are evaluated using a simulator. The evaluations show that these mechanisms can enable more efficient usage of the constrained sensor resources while satisfying the applications requirements for data quality, in spite of sensor dynamics.

The rest of this chapter is organized as follows. Section 4.2 describes the *iZone* programming abstraction and in-network data estimation mechanisms. Section 4.3 presents space, time and resource aware optimizations for in-network interpolation. Section 4.4 presents an experimental evaluation. Finally, Section 4.5 summarizes this chapter.

4.2 In-network Data Estimation

Scientific applications often require data measurements at pre-defined grid points, which are often different from the locations of the raw data provided directly by the sensor network. As a result, a sensor-driven scientific/engineering application requires a virtual layer, where the logical representation of the state of the environment provided to the applications may be different from the raw measurements obtained from the sensor network. The *iZone* abstractions described in this section enables applications to specify such a virtual layer as well as implement the models (e.g., regression models, interpolation functions, etc.) that should be used to estimate data on the virtual layer from sensor readings.

4.2.1 The *iZone* Abstraction

As mentioned above, there is often a mismatch between the discretization of the physical domain used by the application and the physical data measured by the sensor network and as a result, data from the sensors has to be processed before it can be coupled with simulations. The goal of the *iZone* abstraction is to support such an integration. It essentially abstracts away the details of the underlying measurement infrastructure and hides the irregularities in the sensor data by virtualizing the sensor field. The result is a consistent representation over time and space to match what is used by the simulations.

The *iZone* itself is thus a representation of the neighborhood that is used to compute a grid point of the virtual layer, and can be specified using a range of coordinates, functions, etc. The *iZone* abstraction enables the implementation of the estimation mechanisms within the sensor network. For example, interpolation algorithms, such as regressions, inverse distance weighing (IDW), and kriging, require the definition of an interpolation zone, an *iZone*, which defines the neighborhood around the grid point to be estimated, and such neighborhood is then used to compute that interpolation point. Note that for several interpolation algorithms, this zone may change on the fly based on the constraints provided by the application. The *iZone* abstraction provides operators for obtaining sensor measurements corresponding to the region of interest, as well as for defining in-network processing operators to compute a desired grid point from sensor values of this region. The semantics of operators of *discover*, *expand*, *shrink*, *get*, *put* and *aggregate* are discussed in detail with Chapter 3.

Once an *iZone* is defined, computing the data value at a grid point consists of (1) identifying a master node that coordinates the estimation process, which could be the sensor node that is closest to the grid point and has the required capabilities and resources, (2) discovering the sensors in the *iZone* that will be used in the estimation, (3) planning the in-network estimation strategy based on desired cost/accuracy/energy tradeoffs, and (4) performing the estimation and returning the computed data value at the desired grid point.

4.3 STaR: SPace, Time and Resource Aware Optimization

This section explores temporal and spatial correlations in the sensor measurements to reduce costs and handle sensor dynamics, while bounding estimation errors for a given *iZone*. For example, a subset of the sensors in an *iZone* may be sufficient to satisfy the desired error bounds while reducing the energy consumed. Further, temporal regression models can be used to handle transient data unavailability, which may otherwise lead to a significant increase in costs and energy consumption [52].

4.3.1 Saving Energy Using *iSets*

Typically, for a densely deployed sensor network, a subset of sensors across an *iZone* may be sufficient to meet the quality requirements for in-network data estimation. In this case, sensors in the *iZone* are divided into multiple interpolation sets, *iSets*, each of which can be used to estimate the data points while still satisfying the error bounds, and reducing costs and energy consumed. The *iSets* are generated and maintained at runtime to balance cost and energy as well as to tolerate failures.

The problem of generating the *iSets* can be formalized as follows: assume that an *iZone* Z is divided into m exclusive subsets $\{S_1, S_2, ..., S_m\}$ (such that $S_i \cap S_j = \emptyset$ and $S_1 \cup S_2, ..., \cup S_m = Z$), and each subset k ($k = 1, 2, ...m$) contains N_{a_k} number of sensors. The objective is to find "best" collection of *iSets* that satisfies data quality requirements.

The *iSets* should satisfy three requirements: (1) the interpolation error for each *iSet* should be less than the specified error tolerance; (2) the average number of sensor measurements for each *iSet* should be minimized in order to reduce the energy consumed; (3) the average aggregated error (i.e., $1/m \sum_k err(S_k)$) should be minimized in order to achieve data quality whenever possible. In addition to these basic requirements, further constraints may be added to satisfy additional resource consumption and scheduling requirements. For example, the sizes of the *iSets* should be similar to make resource consumption more balanced and the scheduling easier. Similarly, the variance of interpolation errors across the *iSets* should be as small as possible.

The *iZone* is thus divided into m *iSets*, only one *iSet* of which needs to be active at a time. These *iSets* can now be scheduled in a round-robin fashion. Note that, as the number of *iSets* increases (i.e., the average size of *iSets* decreases), the efficiency of the approach increases as well. The generation and maintenance of *iSets* is illustrated in Figure 4.1 and is described below.

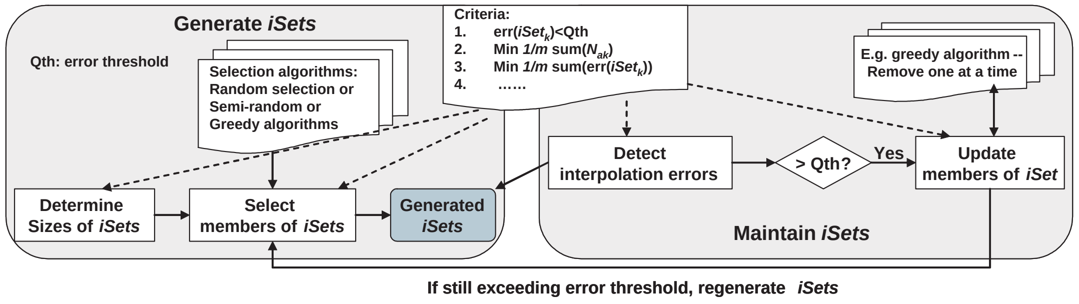

Generate iSets
Qth: error threshold
Selection algorithms: Random selection or Semi-random or Greedy algorithms
Criteria:
1. err(iSet_k)<Qth
2. Min 1/m sum(N_ak)
3. Min 1/m sum(err(iSet_k))
4. ……
E.g. greedy algorithm -- Remove one at a time
Determine Sizes of iSets
Select members of iSets
Generated iSets
Detect interpolation errors
> Qth?
Yes
Update members of iSet
Maintain iSets
If still exceeding error threshold, regenerate iSets

Figure 4.1: An overview of generating and maintaining *iSets*.

4.3.2 Generating *iSets*

Determining the Sizes of *iSets*

The appropriate size of the *iSets* used for interpolation is determined based on the specifications provided by the application, and computed (offline or online) using a stochastic approach as follows. The size of an *iSet* is first initialized to 3 (i.e., $k = 3$). The sensors used for interpolation tests are randomly selected, and the number of tests is set to some reasonable number (i.e., $N_{tr} = 50$). If the interpolation error is within the error tolerance threshold Q_{th}, the successful interpolation counter $succ$ is incremented. If the success rate (i.e., $succ/N_{tr}$) is greater than a specified percentage θ, the algorithm terminates and the current size of the *iSet* is returned as the desired *iSet* size. If the success rate is less than θ after all the tests are completed, the size of *iSet* is incremented and the procedure is repeated until the size equals the size of the *iZone*. Note that this is only done once.

Selecting Members of the *iSets*

Once the size of an *iSet* is determined, the members of each *iSet* are selected so as to satisfy the criteria discussed earlier. A straightforward approach is to use an exhaustive search to find the optimal collection of *iSets*. This approach is obviously expensive for reasonably sized *iZones*, and as a result, we propose approximate algorithms, i.e., the random, semi-random and greedy algorithms, for finding near optimal *iSets*, as described below.

Random algorithm Given N sensors in an *iZone*, this algorithm randomly generates m mutually exclusive *iSets*, each approximately of size k (i.e., $N_{a_k} \approx k$), and $\sum_{k=1}^{m} N_{a_k} = N$. The interpolation error of each *iSet* is then evaluated. If the error for every *iSet* is below the error threshold, Q_{th}, the aggregated error across all the *iSets* is computed and saved. This process is repeated several times. The collection of *iSets* that leads to the smallest aggregated error is finally selected as the initial collection of *iSets*. The number of trials N_{tr} may be explicitly specified or computed based on observed (or historical) data. The actual value depends on the characteristics of sensor data. For example, for the dataset used in the experiments in the paper (see Section 4.4), a suitable value is between 50 and 100.

Semi-random algorithm This algorithm is based on the heuristic that if the sensors in an *iSet* is more uniformly distributed across the *iZone*, the estimation has higher chance to be more accurate. This algorithm attempts to assign neighboring nodes to different *iSets*. This is done using locality preserved space filling curves [99] as the indexing mechanism. First, each sensor is indexed using the Hilbert space filling curve (SFC) based on their locations. Within a given *iZone*, sensors with neighboring identifiers are then virtually grouped based on their SFC indices, so that the size of each virtual group is equal to the number of *iSets* required. For example, if m *iSets* are needed, each group would have m members. The *iSets* are now constructed by selecting one sensor from each of the virtual groups. Note that the selection of sensors from each of the virtual groups is random.

Greedy algorithms Three of greedy algorithms are also devised to select appropriate sensors for the *iSets*. These algorithms are described below.

Greedy Algorithm 1 – "Remove one node at a time": In this algorithm, we start with one *iSet* containing all sensors in the *iZone*. Sensors are then removed one at a time. The removed sensor is the one that leads to minimal interpolation error at each step, until desired *iSet* size is achieved. That is, given the *iSet* k, sensor node j such that

$$j = \arg\min_{i \in S_k} err(S_k - i)$$

is removed from the *iSet*. We then use a random algorithm to select the appropriate members of the last two *iSets*. This is because, if all *iSets* were generated using “remove one at a time”, the last *iSet* would come up together with the $(m-1)$th one without any chance to make any local optimal selection to minimize estimation errors as other *iSets*, and thus usually cannot meet quality requirements. As a result, the last two *iSets* are chosen with another methods, such as random algorithm or the second greedy algorithm.

Greedy Algorithm 2 – “Add one node at a time”: The second algorithm starts with a single sensor node in the initial *iSet*, and adds one node at a time while maintaining the interpolation error constraints. That is, given the *iSet* k, node j is added to the *iSet* to minimize the interpolation error, such that

$$j = \arg\min_{i \in Z \setminus S_k} err(S_k \cup i).$$

This process is repeated until all nodes are assigned to *iSets*.

Greedy Algorithm 3: This algorithm uses the heuristic that nodes which are far from each other are less correlated and as a result are good candidates to add into the existing *iSets*. The idea is to select sensors that far from the last selected sensor. To implement this algorithm, we used the SFC-based indexing method described as part of the semi-random algorithm. Sensor nodes are first indexed using the Hilbert SFC. Virtual groups with sizes equal to number of *iSets* are then formed base on their SFC indices.

The algorithm is initialized by assigning sensors of one virtual group to each *iSet*. Next, sensors are permutated from one of remaining virtual groups, and are mapped to each of m *iSets*. The permutation leading to the least aggregated interpolation error is added to each of them respectively at a time. This step is repeated until all the sensors are assigned to the *iSets*. As stated with the heuristic, the sequence of which virtual group to be added to the *iSets* has the impact on the accuracy of estimations using the *iSets*. As a result, a pre-processing step is used to find such good sequence(s) either online or using historical data. Note that this only needs to be done once.

Once sensors are assigned to *iSets*, interpolation are performed. Next, we describe how to maintain *iSets* when the underlying system changes at runtime.

4.3.3 Maintaining *iSets* at Runtime

Due to the dynamics of underlying physical environment and the sensor system, currently valid *iSets* may not satisfy data quality requirements in the future. As a result, mechanisms are needed to maintain *iSets* to ensure that they continue to meet data quality requirements. In this section, we describe how to maintain *iSets*. We also assume that the sensor network is clustered to construct a two level self-organizing overlay of sensors, in which cluster heads perform coordination of the *iZone*.

Our approach is as follows. First, interpolation errors are tracked by using localized error models at each individual cluster. The error models are localized so that a violation of error thresholds can be detected locally without communicating with other clusters. When a threshold violation is detected, a greedy algorithm is used to update the involved *iSet* to improve estimation quality whenever possible.

Generating models for interpolation errors: It is noted that interpolation errors are often correlated with relevant sensor measurements. As a result, regression models can be used to describe the relationship $e(t) = f(v_k(t))$ between interpolation errors $e(t)$ and the current measurement $v_k(t)$ of sensor k. A combination of offline and online estimation methods can be used to learn such a relationship, in which the coarse trends of error models are learned using offline methods using historical data, while specific local parameters can be learned at runtime. For example, an offline study may suggest that a regression model with degree one, i.e., $a_1 v_k + b_1$, should be used. The model parameters a_1 and b_1 are estimated using previous values of sensor k and the corresponding interpolation errors at runtime at individual cluster heads. These models can then be used to estimate interpolation errors using measurement of sensor k from local cluster.

Maintaining *iSets* using a greedy algorithm: When an *iSet* only temporarily exceeds error thresholds, the greedy "remove one at a time" algorithm can be used at each cluster to temporarily remove sensor measurements from that *iSet*. Each cluster makes recommendation of which node(s) to remove, and the recommendation that provides the least interpolation error is enforced. Note that if the interpolation error still exceeds error threshold, the *iSets* needs to be regenerated.

Transient unavailability using temporal estimation: Since unavailability of scheduled sensors requires re-collection of raw data and thus resulting in expensive communication and energy consumption, temporal models are used to estimate the missing sensor measurement. The idea is to use the fact that neighboring sensor nodes would experience similar changes. As a result, samples from neighboring sensors can be used to facilitate the estimation of temporal model parameters, such as degree of regression model, length of time-series. Note that these parameters would change as the underlying physical characteristics vary. The actual coefficients of temporal model are determined based on previous values of the missing sensor data. The evaluations of these optimized in-network mechanisms are presented next in Section 4.4.

4.4 Experimental Evaluation

In this chapter a simulator is used to evaluate the performance of in-network data estimation mechanisms. The simulator implements the space, time, and resource aware optimization mechanisms for realistic scenarios. The scenarios in the experiments are driven by a real-world sensor dataset obtained from an instrumented oil field with 500 to 2000 sensors, and consisting of pressure measurements sampled 96 time per day. A two-tiered overlay with about 80 clusters is initialized. More powerful nodes are elected as cluster heads and also perform the in-network estimations.

In each experiment, about 500 instances of in-network interpolations are performed on pressure values obtained from simulated sensor nodes. Communication costs are evaluated with and without optimizations. The accuracy and costs are also evaluated in the presence of dynamics of physical environments and sensor systems. Finally, the cost of generating *iSets* is examined using the random, semi-random and greedy algorithms. The primary metrics used in the evaluation are communication cost, measured in terms of number of messages transmitted within the network, and accuracy, measured in terms of relative or absolute interpolation errors.

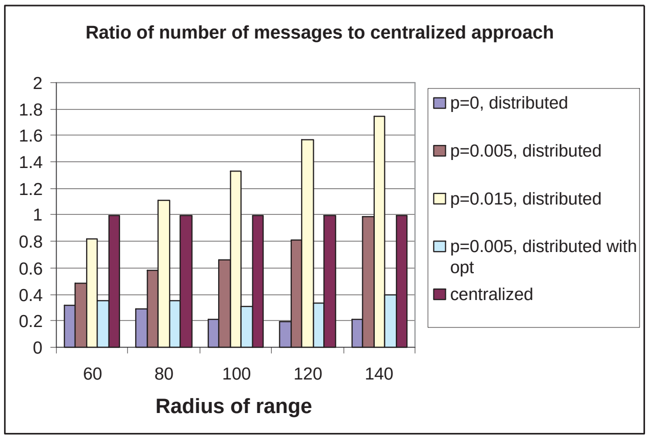


Figure 4.2: Communication cost in the presence of sensor dynamics.

4.4.1 Communication Costs

The current *iZone* prototype implements a distributed in-network mechanism, in which parameters corresponding to the estimation model are first computed at the cluster heads. The cluster heads coordinate the estimation process, and distribute those parameters to the selected *iZone* sensors. A decentralized energy-efficient aggregation scheme is then used to estimate the data. To simulate transient unavailability of sensors, each sensors are given the same unavailability rates, which are the frequencies that scheduled sensors are not available at the time of interpolation. The communication costs are normalized to the cost of the baseline centralized approach, where a coordinator sensor collects raw measurement from selected *iZone* sensors and does the estimation. As plotted in Figure 4.2, the distributed approach performs best when the sensor system is static. With a small unavailability rate of 0.5%, the communication cost increases by over 50% for an *iZone* radius of 60, and over 4 times for a radius of 140. The cost also increases as the unavailability rate increases.

Effectiveness of temporal estimation: In this experiment, the performance of using temporal estimation for temporarily unavailable sensor data is evaluated. Model parameters such as the order of regression functions, and the length of historical data used for the estimation, are chosen at runtime. The estimation parameters varies over time. For example, the length of used historical data is changed from 8 to 6 after interpolation time 6, and the degree of regression models is changed from 3 to 2 after interpolation time 5. The adaptive temporal estimation using spatial-temporal information from neighboring nodes (i.e., circles in Figure 4.3)

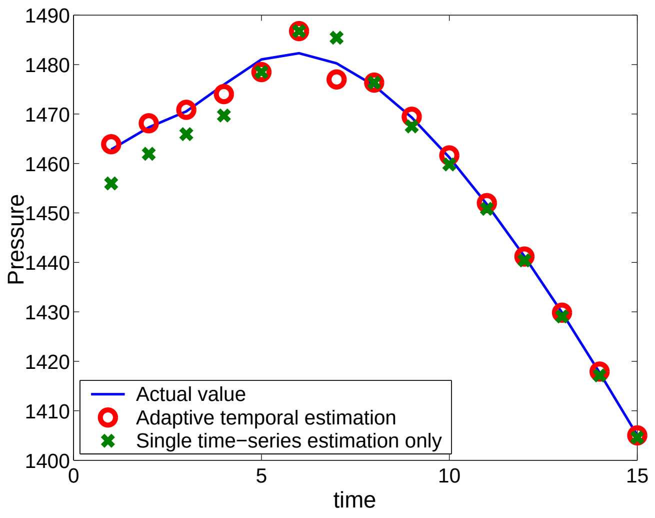


Figure 4.3: Adaptive temporal interpolation.

is much closer to the actual values (i.e., solid line) than that using only historical measurements from sensors with temporally unavailable data (i.e., crosses in Figure 4.3). This is because the spatial-temporal models of neighboring nodes can better catch the changes of the underlying physical characteristics than that only using the model from temporally unavailable sensor. Furthermore, as plotted in Figure 4.2, by using adaptive temporal estimations for temporarily unavailable sensor measurements when possible, the communication cost is reduced by about 25% for a radius of 60, and about 60% for a radius of 140.

4.4.2 Effectiveness of Generating *iSets*

In these experiments, the effectiveness of using random, semi-random and greedy algorithms to generate *iSets* for in-network interpolation tasks are investigated, and three *iSets* are formed within the given *iZones* for this set of experiments.

Effectiveness of random-based algorithms The histograms of average interpolation errors for each of three generated *iSets* are plotted in Figure 4.4 using random and semi-random algorithms respectively. The probability of smaller average interpolation errors using the semi-random algorithm is much higher than that using the random algorithm. For example, for an average interpolation error less than 0.2%, the semi-random algorithms (e.g., with probability about 32%) have higher probability to generate *iSets* meeting quality requirements than that using the random algorithms (e.g., with probability about 12%).

In Figure 4.5, the standard deviation of interpolation errors is examined for the two algorithms. The random algorithm gives much larger variation than the semi-random algorithm. For the random-based algorithms, within the same range of interpolation error (i.e., 0.62%), the standard deviation of random method is still much larger than that of semi-random algorithm. This tells us that the semi-random algorithm is generally more effective in finding the collections of *iSets* having both small interpolation errors and a small variation of such errors.

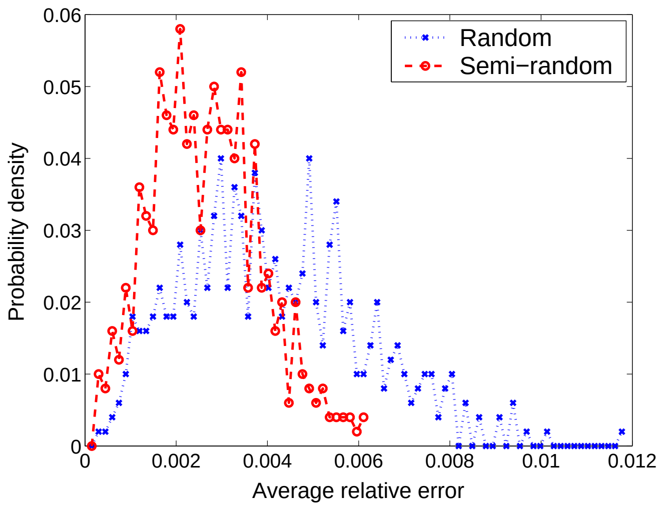


Figure 4.4: Histograms of interpolation errors of generated *iSets*

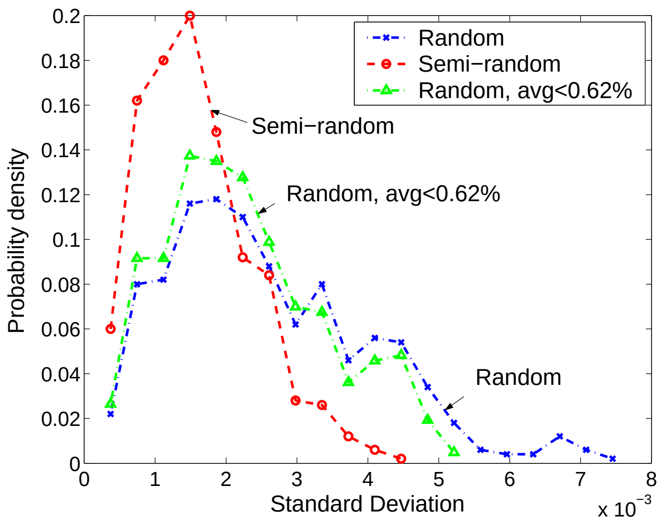


Figure 4.5: Histograms of variation of interpolation errors of generated *iSets*

In Figure 4.6, the number of collections of *iSets* is counted in terms of the absolute average

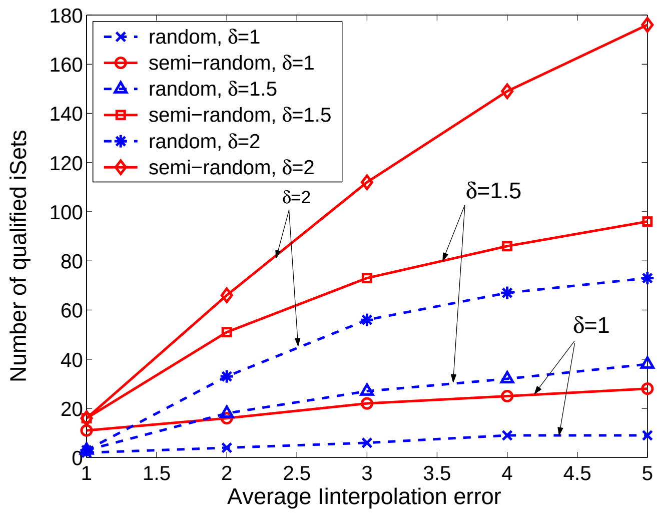


Figure 4.6: Effectiveness of random-based algorithm

interpolation errors and deviations (i.e., δ) among *iSets*. More candidates meeting those requirements are available using semi-random algorithm than that using random algorithm. For example, with small variance 1 (i.e., $\delta = 1$), the available number of collections using semi-random algorithm is five more times than that using random algorithm, which indicates the effectiveness using the semi-random algorithm especially with higher quality requirements.

Effectiveness of greedy algorithms First the three greedy algorithms, as well as random and semi-random algorithms are compared in terms of interpolation errors. Three *iSets* are generated in this example. As shown in Figure 4.7, the first greedy algorithm, "remove one at a time", behaves well for most of the generated *iSets* except the last one. For this algorithm, the last *iSet* exhibits high error since it has no chance to exploit local optimization. The last two *iSets* are thus chosen using other algorithms, such as random algorithms or "add one at a time" greedy algorithm. The second algorithm, "add one at a time" may give relative balanced results, however, the overall error rates could be higher than that of random and semi-random algorithms. The third greedy algorithm gives good accuracy performance comparing to random and semi-random algorithms. The tradeoff is that it needs pre-processing to find good sequence of the next explored sensors. However, the pre-processing could be performed offline and its cost is much less than that of random and semi-random algorithms.

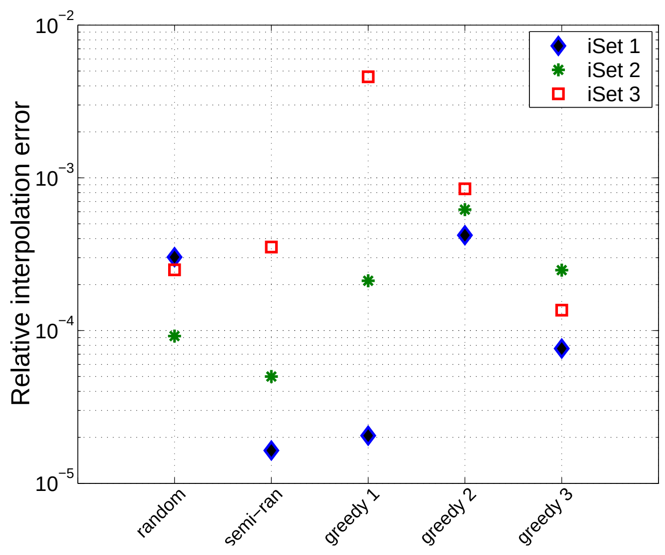


Figure 4.7: Interpolation errors of algorithms generating *iSets*

4.4.3 Tradeoffs between Accuracy and Energy Consumption

In this section, the tradeoff of accuracy and energy consumption is examined. The energy consumption is normalized to one when all sensors in an *iZone* are active at each time. As shown in Figure 4.8, as the sizes of *iSets* becomes smaller, less energy is consumed and the interpolation errors become greater. For example, when half of the nodes are active, approximately half of the energy can be saved, and the range of interpolation errors of selected *iSets* are greater. In addition, the maximum error is slightly greater than when using all nodes. When only one third or one fourth of nodes are active, the maximum error is similar to that of using only half of all nodes. This means that only a portion of active sensors may be able to meet accuracy requirements while saving additional energy. However, when only one fifth of nodes are active, the errors (both maximum and average) become much larger and not meet application requirements anymore.

4.4.4 Cost of Maintaining *iSets*

In this experiment, the communication cost of maintaining *iSets* at local cluster heads is examined. The cost primarily consists of exchanging information, such as calibration information, identifiers of temporarily removed sensors between cluster heads. As shown in Figure 4.9, the cost of maintaining *iSets* at cluster head introduces much less communication overhead

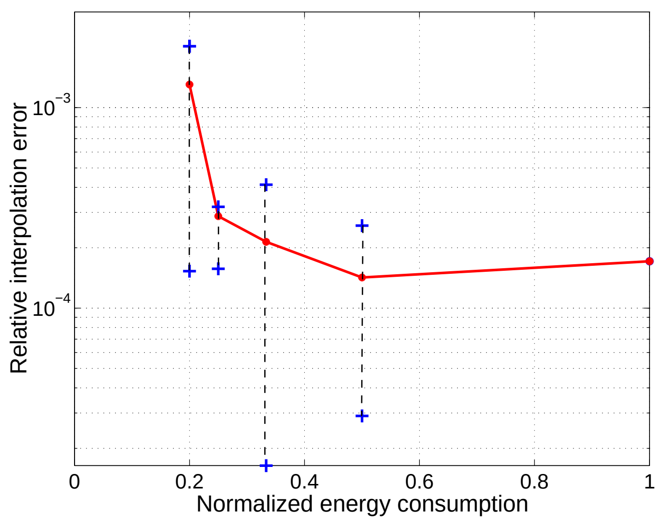


Figure 4.8: Tradeoff between interpolation error vs. energy consumption

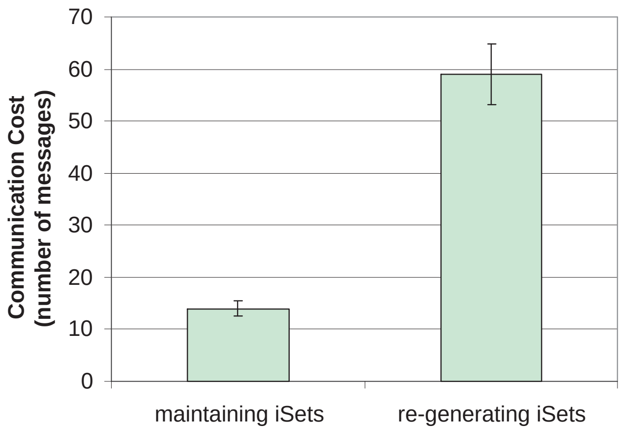


Figure 4.9: Cost of maintaining *iSets*

than regenerating *iSets*. Furthermore, the original error before removing a sensor is $.06069\%$ (threshold is $.06\%$), and after removing one of them using greedy algorithm, the result becomes $.00922\%$, which is far lower than the threshold. This method is quite effective when the change of physical phenomenon is just temporary. After this, the original *iSet* can be used again with the error rate lowered to $.04681\%$. As a result, this method significantly reduced the frequency to re-generate the whole *iSets* in the *iZone*.

4.5 Summary

This chapter investigated abstractions and mechanisms for in-network data processing that can effectively satisfy dynamic data requirements and quality of data and service constraints, as well as investigate tradeoffs between data quality, resource consumptions and performance. Specifically, the proposed mechanisms (i) allow flexibility in the specification of relevant subsets of a sensor network with *iZones* and *iSets*; (ii) explore space, time and resource aware optimizations that utilize the spatial and temporal correlation among sensor measurements to reduce costs while bounding estimations errors; (iii) are robust with respect to network dynamics; and (iv) provide a virtualization of the physical sensor grid to match the representation of the physical domain used by the models, and can dynamically discover and access sensor data independent of any change to the sensor network itself. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of constraint resources and satisfy data requirement in the presence of dynamics and uncertainty.

Chapter 5

A Decentralized Content-based Aggregation Service for Pervasive Grid Environments

This chapter presents a content-based decentralized information aggregation service for pervasive environments. The service provides a uniform query interface where aggregation queries are specified using content descriptors in the form of keywords, partial keywords, wildcards and ranges. The service guarantees that all data that matches a complex/range query will be located and aggregated in an efficient and scalable way. The design of the aggregation service uses a decentralized aggregation trie along with a distributed and decentralized query engine. The deployment and experimental evaluation of the aggregation service are also presented. Evaluations include simulations as well as experiments using deployments on local-area network, the wireless ORBIT testbed [2] at Rutgers and wide-area PlanetLab testbed [93]. Evaluation results demonstrate the scalability, effectiveness and performance of this content-based aggregation service to support pervasive grid applications.

5.1 Introduction

Emerging pervasive information and computational Grids are enabling a new generation of applications that are based on seamless "anytime-anywhere" access to and aggregation of pervasive information, and the interactions between the information sources and distributed services and resources. These applications are context-aware, and use pervasive information about the environment and user's preferences and actions to tailor services and applications to the user's needs, and to automate tasks in a transparent manner. Applications range from everyday activities (applications that use sensors to monitor and manage an office building or a home) to emergencies and crisis management (response to an accident or fighting a fire).

Illustrative scenarios that leverage pervasive environments integrating sensor/actuator devices with distributed services and resources include scientific/engineering applications that symbiotically and opportunistically combine computations, experiments, observations, and real-time data to manage and optimize its objectives (e.g. oil production, weather prediction), pervasive applications that leverage the pervasive/ubiquitous information Grid to continuously manage, adapt, and optimize our living context (e.g. your clock estimates drive time to your next appointment based on current traffic/weather and warns you appropriately), crisis management applications that use pervasive conventional and unconventional information for crisis prevention and response, medical applications that use in-vivo and in-vitro sensors and actuators for patient management, ad hoc distributed control systems for automated highway systems, manufacturing systems or unmanned airborne vehicles, and business applications that use pervasive information access to optimize profits.

A key application driving this research is the sensor-driven management of subsurface geosystems and specifically the dynamic data-driven management and optimization of oil reservoirs [61,82]. In this application sensor data is used dynamically and opportunistically to detect suboptimal or anomalous behavior, by optimization-based strategies for parameter estimation, and to provide initial conditions to dynamically adaptive forward simulation models.

Another potential class of application is crisis management. For example, one can conceive of a fire management application where computational models use streaming information from sensors embedded in the building along with real time and predicted weather information (temperature, wind speed and direction, humidity) and archived history data to predict the spread of the fire and to guide firefighters, warning of potential threats (blowback if a door is opened) and indicating most effective options. This information can also be used to control actuators in the building to manage the fire and reduce damage.

While recent technical advances and cost dynamics in computing and communication technologies are rapidly enabling the realization of the pervasive grid computing vision, these environments and applications continue to present several significant challenges. In addition to the challenges of distribution, large scale and system heterogeneity, pervasive computing environments are inherently dynamic. For example, sensors are often resource/energy limited and mobile, and may dynamically join, leave, or fail. As a result pervasive applications must adapt to the unreliability and uncertainty of information and services. Further, interactions between devices, services and resources in a pervasive application are also dynamic and typically ad-hoc and opportunistic. Similarly, due to the heterogeneity and complexity of phenomena being monitored and modeled, applications may require different aggregations in different regions of the domain.

Addressing the issues outlined above requires a middleware infrastructure that can effectively extract and abstract desired information from the huge amounts of data present in sensor-based systems, despite their scale, heterogeneity and dynamism. Further, it should automate some of the decision processes associated with the computation in the network. This chapter presents a decentralized content-based aggregation service to support applications with heterogeneous in-network aggregation requirements in dynamic and widely distributed pervasive environments. Key characteristics of the aggregation service are as follows. (1) It seamlessly integrates pervasive information sources (e.g., sensor devices) with networked services, resources and applications; (2) A *uniform* query interface where aggregation queries are specified using content descriptors and an action descriptor. The content descriptors may be keywords, partial keywords, wildcards and ranges derived from a semantic space, and the action descriptor define aggregation operations; (3) The service guarantees that all data matching a complex/range query will be located and efficiently and scalably aggregated.

The aggregation service builds on the Meteor content-based middleware infrastructure and extends the Associative Rendezvous (AR) model for content-based information discovery and decoupled interactions. Specifically, it extends the AR abstractions to enable content-based aggregation queries to be flexibly specified using keywords, partial keywords and ranges. This also includes specification of recurrent and spatially constrained queries. Further, it builds on a self-organizing overlay network and the Squid content-based routing infrastructure to construct aggregation tries so that query propagation routes can be used for back-propagating and aggregating matching data elements. The design, implementation and evaluation of the aggregation services are presented. Evaluations include experiments using deployments on a local area network, a wireless network testbed ORBIT [2], the wide-area PlanetLab testbed [93], as well as simulations of systems with thousands of nodes.

The rest of the chapter is structured as follows. Section 5.2 reviews the associative rendezvous (AR) model and describes the abstractions and semantics of the content-based aggregation service using the AR model. Section 5.3 introduces the Meteor middleware infrastructure and presents the design of the aggregation service. Section 5.4 presents the experimental evaluation. Section 5.5 concludes the chapter.

5.2 The Content-based Aggregation Service

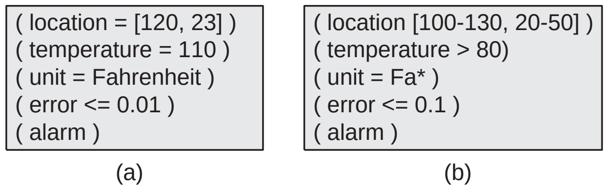
The content-based aggregation service provides a uniform query interface where aggregation queries are specified using content descriptors from a semantic space in the form of keywords, partial keywords, wildcards and ranges, and the type of aggregation desired. Spatial attributes of a data item, e.g., the location of the sensor producing the data can be specified along with the other attributes in a query. The service extends the Associative Rendezvous (AR) [55] abstraction for content-based information discovery and decoupled interactions. In this section, we first introduce the associative rendezvous model and then describe the semantics of the aggregation service.

5.2.1 Associative Rendezvous Messaging

The AR interaction model [55,56] consists of three elements: *Messages*, *Associative Selection*, and *Reactive Behaviors*, which are described below.

AR Messages: An AR message is defined as the triplet: $(header, action, data)$. The data field may be empty or may contain the message payload. The header includes a semantic profile in addition to the credentials of the sender, a message context and the *TTL* (time-to-live) of the message. The profile is a set of attributes and/or attribute-value pairs, and defines the recipients of the message. The attribute field must be a keyword from a defined information space while the value field may be a keyword, partial keyword, wildcard, or range from the same space. At the Rendezvous Point (RP), a profile is classified as a *data profile* or an *interest profile* depending on the action field of the message. The *action* field of the AR message defines the reactive behavior at the RP as described below.

Associative Selection: Profiles in AR are represented using a hierarchical schema that can



(location = [120, 23])
(temperature = 110)
(unit = Fahrenheit)
(error <= 0.01)
(alarm)

(a)

(location [100-130, 20-50])
(temperature > 80)
(unit = Fa*)
(error <= 0.1)
(alarm)

(b)

Figure 5.1: Sample message profiles: (a) a data profile for a temperature sensor; (b) an interest profile for a client.

Actions	Semantics
store	Store data profile and data in the system at the RPs. Match the data profile with existing interest profiles with “notify_data” action. Execute action associated with a matched profile.
retrieve	Match interest profile with existing data profiles; Send data associated with the matched profiles to the requester.
notify_data	Store the interest profile and the action in the system at the RPs. Match the interest profile with: 1) existing data profiles, and send back a notification if a match occurs. and 2) existing interest profiles with “notify_interest” action, Send a notification to the data producer if a match occurs.
notify_interest	Store the interest profile and the action in the system at the RP. Match the interest profile with existing interest profiles with “notify_data” action. Send a notification to the data producer if a match occurs.
delete_data	Match the profile with existing data profiles. Delete all matching data profiles with appropriate credentials, and the data associated with them.
delete_interest	Match the interest profile with existing interest profiles. Delete all matching interest profiles with appropriate credentials.

Table 5.1: Basic reactive behaviors.

be efficiently stored and evaluated by the selection engine at runtime. For example, the profile in Figure 5.1 (a) is associatively selected by the profile in Figure 5.1 (b), since (1) the location [120, 23] of the data is within the region of interest defined by the ranges [100-130, 20-50], (2) the attribute temperature matches and its value $100 > 80$, which satisfies the binary relation, (3) the attribute unit matches, and its value Fahrenheit matches wildcard $Fa*$, (4) $error < 0.01$ satisfies the request error¡0.1, and (5) the attribute alarm matches. A key characteristic of the selection process is that it does not differentiate between interest and data profiles. This allows all messages to be symmetric where data profiles can trigger the reactive behaviors of interest messages and vice versa. The matching system combines selective information dissemination with reactive behaviors. Further, both data and interest message are persistent, with their persistence defined by the *TTL* field.

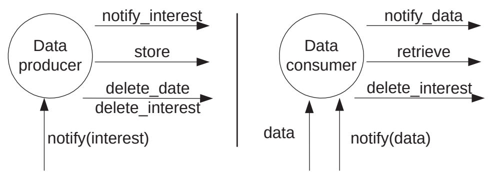


Figure 5.2: Basic reactive behaviors: (a) for message issued by data producers; (b) for messages issued by data consumers.

Reactive Behaviors: The *action* field of the message defines the reactive behavior at the rendezvous point. Basic reactive behaviors currently defined include *store*, *retrieve*, *notify*, and *delete* as listed in Table 5.1. These reactive behaviors are used by data producers and consumers to store/retrieve data as illustrated in Figure 5.2. Note that a client in the system can be a data producer, a data consumer, or both.

The *notify* and *delete* actions are explicitly invoked on a data or an interest profile. The *store* action stores the data and data profile at the rendezvous point. It also causes the message profile to be matched against existing interest profiles with *notify_data* action, and the data to be sent to the data consumers that requested it in case of a positive match.

The *retrieve* action retrieves data corresponding to each matching data profile. The *notify* action matches the message profile against existing interest/data profile, and notifies the sender if there is at least one positive match. The *notify* action comes in two flavors: *notify_data* and *notify_interest*. *Notify_data* is used by data consumers, who want to be notified when data matching their interest profile is stored in the system. *Notify_interest* can be used by data producers, who want to be notified when there is interest in the data they produce, so they can start sending data into the system.

Finally, the *delete* action deletes all matching interest/data profiles. Note that the actions will only be executed if the message header contains an appropriate credential. Also note that each message is stored at the rendezvous for a period corresponding to the lifetime defined in its header. In case of multiple matches, the profiles matching are processed in random order.

5.2.2 Opportunistic Application Flows in Pervasive Environments

In this section, we describe the Cascading Local Behavior (CLB) programming model, in which the behaviors of individual application elements (i.e., sensors, actuators, services) are locally defined in terms of local state, and context and content events, and result in data and interest messages being produced. Interactions, compositions and application flows emerge as a consequence of the cascading effect of such local behaviors, without having to be explicitly programmed.

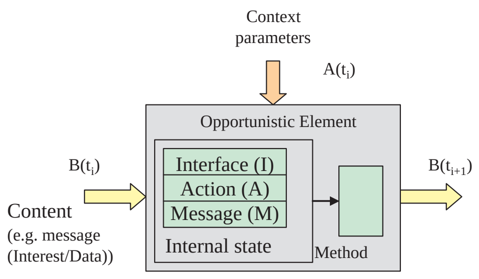


Figure 5.3: Defining local behaviors.

Cascading Local Behaviors (CLB) is a model for realizing opportunistic applications flows in pervasive sensor/actuator-based environments. It builds on a common semantic basis (i.e., ontology and taxonomy) for describing content and context. In the CLB model only the local behaviors for each element (i.e., sensor, actuator, resources, services) are programmed. These behaviors can be viewed as a state machine where the local state is defined in terms of a local actions (A), active interfaces (I) and active data and/or interest messages (M), as illustrated in Figure 5.3. Local state transitions are triggered by context and/or content events and may results in local actions (e.g., update database or turn on an indicator) and the generation of data/interest messages. Note that the definition of the local behavior are independent of the rest of the pervasive system. Now, messages generated during local state transitions may trigger transitions in other element, which in turn may generate further messages. The resulting cascading local transitions cause applications flows to opportunistically emerge.

For example, consider very simple smart home scenario with device sensors and actuators shown in Figure 5.4. The local behaviors of these sensors/acutators are illustrated as `if-then` rules in the figure. The temperature sensor monitors local temperature and generates a *post(temp = 91, store)* data message when the temperature rises above some threshold. This causes AR to generate a notification to the thermostat actuator which then turns the air conditioning on and generates a *temp-control* message. Now other sensors/actutors that are subscribed will be notified and can react, for example, a window could shut itself or a fan could turn itself off. The devices need not know other each other as long as they use a common semantic basis for describing content and context.

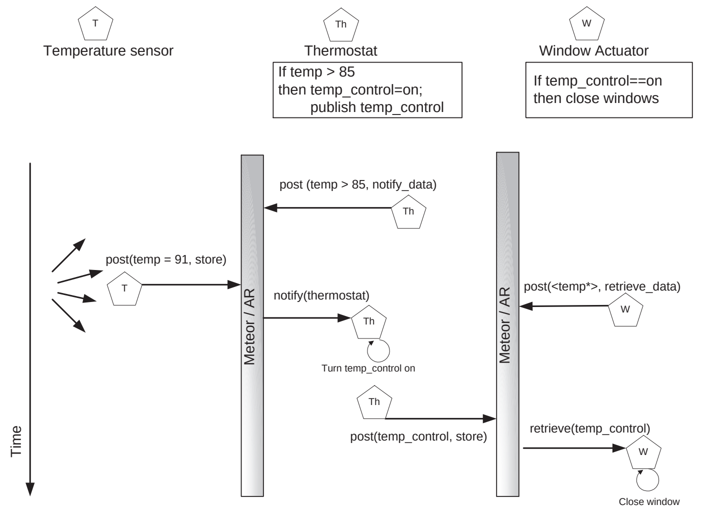


Figure 5.4: Cascading local behaviors - an illustrative example.

Note that CLB differs from traditional composition-based programming approaches, where the desired end-to-end behavior is known a priori and is used to define the local behaviors of the elements as well as their interactions. In CLB, the behaviors of elements can be independently defined without knowledge of the functionality or existence of other elements. Further, elements in the system can spatially and temporally decoupled, i.e., an element entering the system at a later time (for example, the window actuator in Figure 5.4) can still be part of an emerging flow. A key requirement for CLB is a communication/interaction infrastructure that: (a) is scalable and self-managing, (b) is based on content rather than names and/or addresses, (c) supports asynchronous and decoupled interactions rather than forcing synchronization, and (d) provides some interaction guarantees. Such an abstraction is provided by AR.

5.2.3 Information Aggregation using Associative Rendezvous

The aggregation service extends the *retrieve* reactive behavior to specify aggregation operators, i.e., the AR message for an interest profile includes the aggregation operation as follows: $(< \{\texttt{attr}\} >, retrieve(A))$, where `attr` is the set of content attributes of the interest profile within the AR message header, and $retrieve(A)$ specifies aggregation using the aggregation operator A. Note that aggregations can be constrained to data-elements within a specific spatial region. The location attributes of a data element for an aggregation query are specified as part of the semantic profile of the message header, similar to other content attributes, i.e., location descriptors form (typically leading) dimensions of the semantic information space, based on which data and interest profiles are defined. The data profile becomes $(< L, \{\texttt{attr}\} >, store, data)$ and the corresponding interest profile becomes $(< L, \{\texttt{attr}\} >, retrieve(A))$, where *L* specifies the location of the data producer in the data profile and the region of interest in the interest profile. *L* may use latitude/longitude or any other specification of location. For example, in Figure 5.1, the data profile includes the location of the sensors, while the interest profile includes ranges specifying the region of interest. Aggregation queries may also be recurrent in time. In such a query, an additional parameter is required to specify the frequency of evaluation of the persistent query; $(< L, \{\texttt{attr}\}, TTL >, retrieve(A, T_a))$, where T_a is the time interval between repeated aggregates.

Semantics of Aggregation: The semantics of the aggregation query are as follows. When an aggregation query is posted, the query is routed to all rendezvous peers with data profiles that match the query's interest profile. All data items at all peers that match the interest profile specification are aggregated according to the specified aggregation operator and are returned. In case of recurrent aggregation queries, the interest profile is registered at each rendezvous peer for the duration of its *TTL*, and is repeatedly evaluated at the specified frequency. The aggregation operation is repeated each time the query is evaluated and the aggregate is returned. Note that each aggregation operation is independent.

An Illustrative Example: To illustrate the operation of the aggregation service, consider a traffic monitoring system with deployed vehicle speed sensors. An example of an aggregate query for such a system is *find the average speed in the stretch of road specified by region L every 5 minutes for the next 1 hour*. An aggregation query would be realized using the aggregation service described above as follows: the client connects to any rendezvous peer in the system and posts an aggregate query with profile $< L, p_1 >$, and retrieves information using the aggregator *AVG* defined over region L, with an aggregation frequency of 5 minutes and a *TTL* of 1 hour. Such an aggregate query can be written as *post*$(< L, p_1, 3600 >, retrieve(AVG, 300))$, where the time is measured in seconds. This query is routed to, and registered at every peer that stores data elements matching this query. In response, every matching data-elements in the system are aggregated and returned to the client.

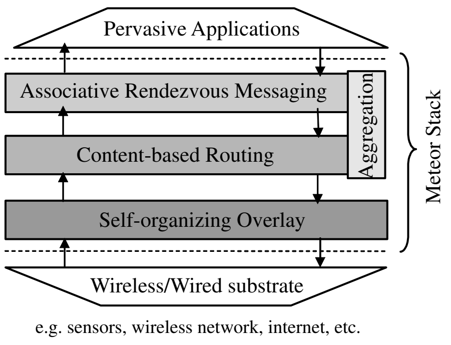


Figure 5.5: Meteor stack - schematic overview.

5.3 Decentralize In-network Aggregation

Meteor is a middleware infrastructure for content-based decoupled interactions in pervasive environments. It is essentially a peer-to-peer network of Rendezvous Points (RP), where each RP is a peer and may be a broadband access point, a forwarding node in a sensor network or a server node in a wired network. To use Meteor, applications must connect to a RP. A schematic overview of the Meteor stack is presented in Figure 3.2. It consists of 3 key components: (1) a self-organizing overlay, (2) a content-based routing infrastructure (Squid), and (3) the Associative Rendezvous Messaging Substrate (ARMS). The aggregation service [57] specifically builds on the Squid CBR infrastructure to construct aggregation trie structures on top of the routes used by the queries, and use them to back-propagate matching data while performing aggregations at intermediate nodes in the trie. Further, it extends the ARMS layer to provide a unified content-based abstraction to specify aggregation operations as reactive behaviors of content-based queries.

5.3.1 The Overlay Network Layer

The Meteor overlay network is a one-dimensional self-organizing structured overlay composed of RP nodes. Peers in the overlay can join or leave the network at any time. While the design of Meteor is not tied to any specific overlay topology, the current Meteor prototype builds

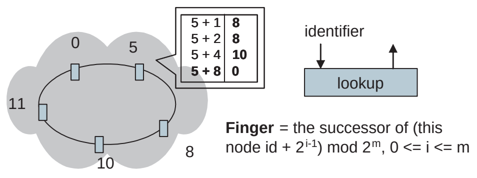

0
5
11
10
8
5 + 1 | 8
5 + 2 | 8
5 + 4 | 10
5 + 8 | 0
identifier
lookup
Finger = the successor of (this node id + 2^{i-1}) mod 2^m, 0 <= i <= m

Figure 5.6: The Meteor overlay network layer.

on Chord [108]. Advantages of Chord include its guaranteed performance, logarithmic in the number of messages (e.g. data lookup requires O($\log N$) messages, where N is the number of RP nodes in the system). However, this overlay could be replaced by other structured overlays.

Peer nodes in the Chord overlay form a ring topology. Every node in the Chord overlay is assigned a unique identifier ranging from 0 to $2^m - 1$. Each data item stored in the system is associated with a key and mapped to an identifier from the same interval. The identifiers are arranged as a circle modulo 2^m. Each node stores the keys that map to the segment of the curve between itself and its predecessor node. Figure 5.6 shows an example of a Chord overlay network with five nodes and an identifier space from 0 to 64.

The overlay network layer provides a simple abstraction to the layers above, consisting of a single operation: **lookup**(*identifier*). Given a numerical identifier, the node responsible for it will be located in $O(\log N)$ hops, where N is the number of nodes in the system.

5.3.2 Content-based Routing Layer

Squid builds on top of the Chord overlay to enable flexible content-based routing. As mentioned above, the **lookup** operator provided by the Chord overlay requires an exact identifier. Squid effectively maps complex message profiles, consisting of keyword tuples made up of complete keywords, partial keywords, wildcards, and ranges, onto clusters of identifiers. It guarantees that all peers responsible for identifiers in these clusters will be found with bounded costs in terms of the number of messages and the number of intermediate RP nodes involved.

Tuples of d keywords, wildcards, and/or ranges represent points or regions in a d-dimensional information space. A point corresponds to a keyword tuple that contains only complete keywords, and is called *simple*, as shown in Figure 5.7 (a). If a tuple contains partial keywords, wildcards and/or ranges it is called *complex* and defines a region in the information space, as shown in Figure 5.8 (a).

Squid uses the Hilbert Space Filling Curve (SFC) [99] to map the multidimensional information space to the 1-dimensional identifier space of the peer overlay. Figure 5.7 (b) shows an example of Hilbert SFC for a 2-dimensional space. The Hilbert SFC is a locality preserving continuous and recursive mapping from a k-dimensional space to a 1-dimensional space. It is locality preserving in that points close on the curve are mapped from close points in the k-dimensional space. The Hilbert curve readily extends to any number of dimensions, though for practical purposes its locality preserving property can best be exploited with under 6 dimensions. Further, its locality preserving and recursive nature enables the index space to maintain content locality and efficiently resolve content-based lookups [101].

Content-based routing in Squid is achieved as follows: SFCs are used to generate the 1-dimensional index space from the multi-dimensional keyword space. Applying the Hilbert mapping to this multi-dimensional space, each profile consisting of a simple keyword tuple can be mapped to a point on the SFC. Further, any complex keyword tuple can be mapped to regions in the keyword space and the corresponding clusters (segments of the curve) in the SFC (see Figure 5.8 (a) and (b)). The 1-dimensional index space generated corresponds to the 1-dimensional identifier space used by the Chord overlay. Thus, using this mapping, RP nodes corresponding to any simple or complex keyword tuple can be located. The Squid layer of the Meteor stack provides a simple abstraction to the layer above consisting of a single operation: **deliver**(*keyword tuple, data*), where data is the message payload provided by the messaging layer above. The routing process is described below.

Routing using Simple Keyword Tuples

The routing process for a simple keyword tuple is illustrated in Figure 5.7. It consists of two steps: first, the SFC-mapping is used to construct the index of the destination peer node from the simple keyword tuple, and then, the overlay network lookup mechanism is used to route to the appropriate peer node in the overlay.

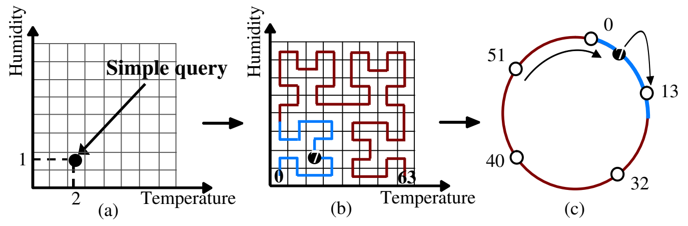


Figure 5.7: Routing using a simple query (a) query (2, 1) is a point in a 2D space; (b) the query is mapped to SFC index 7; (c) the data is routed to peer 13 (in an overlay with 5 peer nodes and an identifier space from 0 to 2^6-1), the successor of the index 7.

Routing using Complex Keyword Tuples

The complex keyword tuple identifies a region in the keyword space, which in turn corresponds to clusters of points in the index space. For example, in Figure 5.8 (a), the complex keyword tuple (2-3, 1-5) representing data read by a sensor (temperature between 2 and 3 units and humidity between 1 and 5 units) identifies 2 clusters with 6 and 4 points respectively.

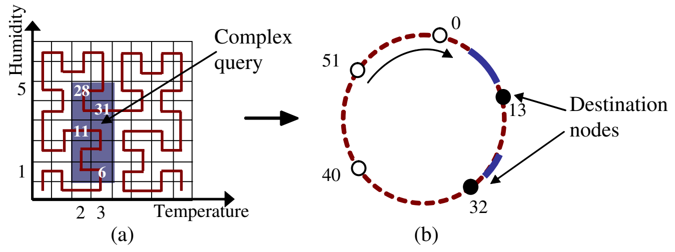


Figure 5.8: Routing using complex query (2-3, 1-5): (a) the query defines a rectangular region in the 2D space, and 2 clusters on the SFC curve; (b) the clusters (the solid part of the circle) are stored at peers 13 and 32.

Thus a complex keyword tuple is mapped by the Hilbert SFC to clusters of SFC indices and correspondingly, multiple destination identifiers. Once the clusters associated with the complex keyword tuple are identified, a straightforward approach consists of using the overlay lookup mechanism to route individually to each RP node. Figure 5.8 (b) illustrates this process. However, as the originating RP node cannot know how the clusters are distributed in the overlay network, the above approach can lead to inefficiencies and redundant messages, especially when there are a large number of clusters. The routing process can be optimized by using the recursive nature of the SFC to distribute the list of clusters to be routed. This optimization is presented in detail as trie construction and trie-based routing as follows.

Trie Construction

Since SFCs are infinitely self-similar recursive data structures, their construction process can be regarded as a tree. Because of their *digital causality* property [1], the tree is a prefix tree, i.e., a trie. A query defines a sub-space in the multidimensional space and segments (called clusters) on the SFC curve. The construction of these clusters corresponds to the construction of a trie.

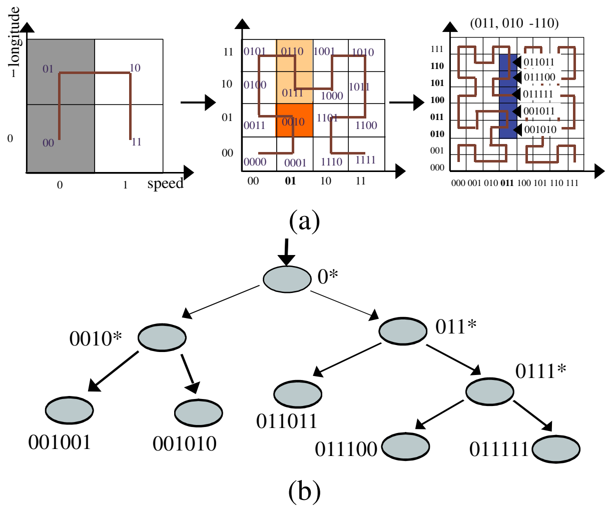


Figure 5.9: Trie construction: (a) the complex query (011, 010-110) - the first, second and third SFC refinement; (b) the trie associated with the query.

Figure 5.9 (a) illustrates the recursive resolutions of the query (011, 010-110) in a 2-dimensional keyword space, with base-2 digits as coordinates. Figure 5.9 (b) shows the construction of the corresponding trie. At each recursion step the discretized space is refined, resulting in a longer curve. The query defines 2 points on the first SFC refinement (prefix 0); 3 points on the second SFC refinement, grouped based on prefixes 011 and 0010; and 5 points on the third SFC refinement, with prefixes 011, 0111 and 0010.

Trie-based Routing

The trie constructed by resolving the query is embedded into the overlay as follows. Each node in the trie is mapped to a peer node in the overlay based on its identifier. The leaf nodes have the SFC index as their identifiers. For intermediate nodes, the identifier is constructed by padding

[1]SFC digital causality: Each step of recursion transforms a point on the SFC curve in multiple points, by extending its identifier with d digits, where d is the dimensionality of the space mapped by the curve.

the prefix with zeroes until the maximum bit-length is reached (i.e. the length that corresponds to the SFC index at the maximum level of refinement allowed). The peers responsible for the identifiers are then located using the lookup mechanism provided by the overlay network.

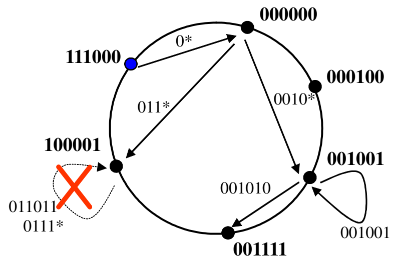


Figure 5.10: Embedding the tree from Figure 5.9 (b) into the overlay network. Node 100001 is responsible for storing the subtree routed at 011*.

Figure 5.10 illustrates the process. In the figure, the overlay network uses an identifier space from 0 to 2^6, and binary node identifiers. The source node, 111000, refines the query at the first recursion level, which results in a cluster with prefix 0. The node then uses the prefix 0 to construct an identifier, by padding the prefix with zeroes, and sends a message to node 000000. At node 000000, the query is refined to generate the second level of recursion, which result in two clusters, one with prefix 011 and the other with prefix 0010. Node 000000 constructs a sub-query identifier for each cluster, and sends the messages to appropriate nodes in the overlay. The nodes that receive the sub-queries refine them to generate the next level of recursion, and so on. Note that node 100001 does not need to refine its sub-query since the sub-query prefix, 011, is smaller than the node's prefix of the same length, i.e., 100, meaning that the entire subtree routed at 011* is stored at this node. As a result, entire sub-trees of the trie associated with the query can be pruned, saving communication and computational resources.

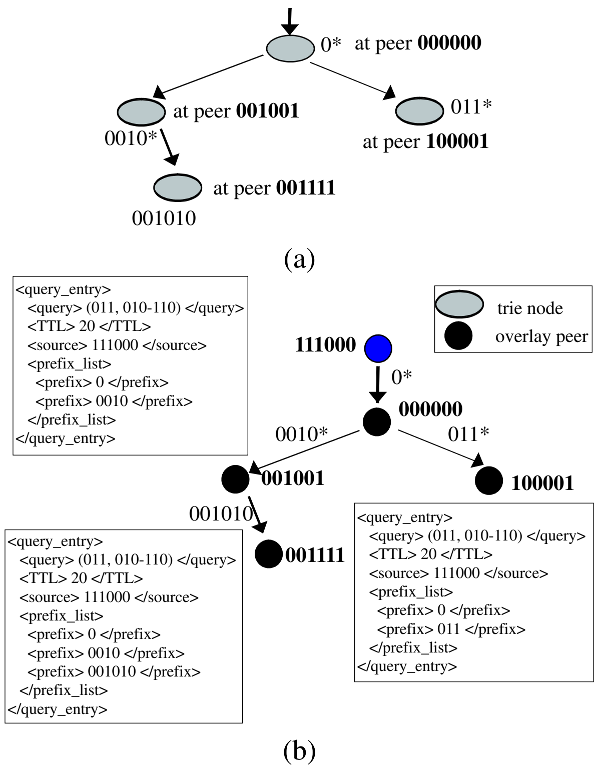
The example presented above has been simplified for ease of illustration. In reality each node performs multiple query refinements (e.g. 5), which result in a large set of sub-queries, each with a longer prefix.

5.3.3 Trie-based In-Network Aggregation

The design of the aggregation service builds on the trie structure constructed by the routing engine. An aggregation query is routed to the appropriate peer nodes with data that matches the query in the usual way. Further, recurrent aggregation queries are stored at the peer node for the duration of their *TTL*s and are periodically evaluated based on the specified frequency. Conceptually aggregation consists of propagating data matching the query back up the trie, with partial aggregation performed at intermediate peer nodes and the final aggregate being computed at the peer node that issued the query.

A straightforward implementation of the service, which assumes that the overlay is static and stable with no peer nodes joining, leaving or failing, consists of maintaining state about each query (e.g., the query operation, the parent and children node for the associated trie) at each peer node that forms the trie. The leaf nodes of the trie evaluate the query, perform the first aggregation, and send the result to their parent peer nodes. Each intermediate node wait until all their children report their partial aggregates, aggregate these results, and forward them to their parents. However such a simple implementation has two problem. First, the overlay is typically dynamic with peers joining, leaving and failing relatively often, and maintaining state about queries can be expensive or infeasible. Second, the number of aggregation queries can be very large and storing state for each query will require significant resources. The approach presented in this chapter does not require explicitly storing query state at the peer nodes. Instead, the prefixes of the nodes in the path of the query along the trie are maintained within the query itself, and the query is only stored at the leaf nodes in case of a recurrent query. This list of prefixes can then be used to back propagate and aggregate data to the source peer node - each peer uses the prefix that precedes its own in the list to route to its parent in the trie.

To illustrate the process, consider the query (011, 010-110) presented in Section 5.3.2. The trie associated with this query was presented in Figure 5.9. Further, as show in Figure 5.10, only a part of the trie is actually constructed while resolving the query. The constructed trie is shown in Figure 5.11 (a). Figure 5.11 (b) shows the list of prefixes accumulated by the query as it reaches different leaf nodes of the query trie. The in-network aggregation process is show in Figure 5.12, and the pseudocode for the aggregation algorithm is presented in Figure 5.13. The leaf nodes evaluate the query and locally aggregate matching data elements. These partial aggregates are then sent to the overlay peer responsible for the parent node in the query trie. This peer is located using the prefix of the parent trie node, by padding it with zeroes to obtain a valid identifier, and performing a lookup for this identifier. Note that the lead node includes the



```
<query_entry>
  <query> (011, 010-110) </query>
  <TTL> 20 </TTL>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
    <prefix> 0010 </prefix>
  </prefix_list>
</query_entry>
```

```
<query_entry>
  <query> (011, 010-110) </query>
  <TTL> 20 </TTL>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
    <prefix> 0010 </prefix>
    <prefix> 001010 </prefix>
  </prefix_list>
</query_entry>
```

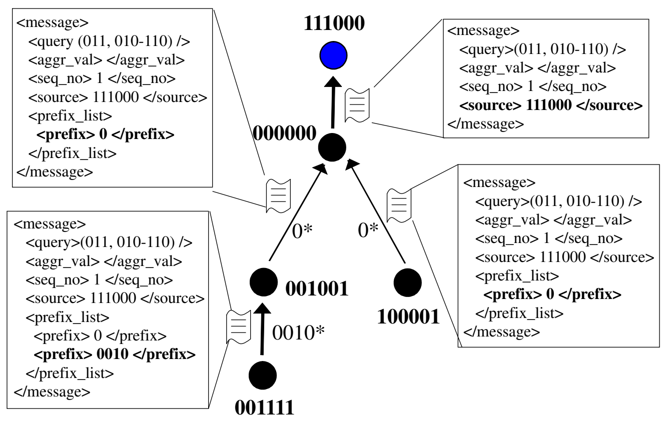
```
<query_entry>
  <query> (011, 010-110) </query>
  <TTL> 20 </TTL>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
    <prefix> 011 </prefix>
  </prefix_list>
</query_entry>
```

Figure 5.11: (a) The trie constructed while resolving the query (011, 010-110) as presented in Figure 5.10; (b) The list of prefixes accumulated by the query at the leaf peer node.

prefix list in the result message. Each intermediate node in the query trie aggregates all partial results it receives for a query form its children in the query trie and periodically forwards these to its parent in the query trie using the same process. Every time a peer node forwards a result up the trie, it remove its prefix from the prefix list.

Dealing with varied link latencies

Each intermediate peer node in the trie that forwards a query also gathers the partial aggregates from its children peer nodes in the trie. This node waits for a predefined time interval, aggregates all partial aggregates that it has received for a query during the interval and forwards them to its parent peer node in the trie. However, since link latencies between peer nodes can vary significantly, it is very difficult, if not impossible, to chose the right duration of the time interval. To address this issue, each partial aggregation message is stamped with a sequence number. Messages associated with the same query and the same sequence number are aggregated. Each intermediate peer node registers the first partial aggregate message it receives in a



```
<message>
  <query (011, 010-110) />
  <aggr_val> </aggr_val>
  <seq_no> 1 </seq_no>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
  </prefix_list>
</message>
```

```
<message>
  <query>(011, 010-110) />
  <aggr_val> </aggr_val>
  <seq_no> 1 </seq_no>
  <source> 111000 </source>
</message>
```

```
<message>
  <query>(011, 010-110) />
  <aggr_val> </aggr_val>
  <seq_no> 1 </seq_no>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
    <prefix> 0010 </prefix>
  </prefix_list>
</message>
```

```
<message>
  <query>(011, 010-110) />
  <aggr_val> </aggr_val>
  <seq_no> 1 </seq_no>
  <source> 111000 </source>
  <prefix_list>
    <prefix> 0 </prefix>
  </prefix_list>
</message>
```

Figure 5.12: In-network, trie-based aggregation. Each peer node performs a partial aggregation, and sends the result to the peer node corresponding to its parent in the query trie.

temporary table, and waits for time *T* for other partial aggregates to arrive. A new aggregation is computed, and a message is sent to the next prefix corresponding to its parent in the query trie, which should be common to all the messages aggregated. If additional messages for the same query and with the same sequence number arrive after time *T*, the process is repeated. The source peer node will perform the final aggregation.

```
if(thisNode is a trie leaf node)
    results = evaluateQuery(q);  //evaluate query q
    aggrVal = aggregate(results);  //constructs the aggregate
    prefixListMsg = q.prefixList;  //copy the prefix list
    q.sequenceNumber = q.sequenceNumber + 1;
else  //this peer is an intermediate node in the trie
    //wait for the chidren nodes to report partial aggregates
    while (waitTime not expired)
        aggrMsg = receive();
        aggrMsgList.add(aggrMsg);
    aggrVal = aggregate(aggrMsgList);
    //the prefix lists are identical for all children
    prefixListMsg = aggrMsg.getPrefixList();

prefixListMsg.removeLast();   //remove the prefix of this node
//retrieves the prefix of the parent node
parentPrefix = prefixListMsg().getLast();
message = constructMessage(q, aggrVal, q.sequenceNumber,
      q.sourceNode, prefixListMsg);
if (parentPrefix != null)   //the prefix list is not empty
    parentID = constructID(parentPrefix);
    parentPeer = lookup(parentID);
    //send the message to the parent peer
    sendMessage(parentPeer, message);
else //this is the root of the trie (or one of them)
    //send the message to the node that issued the query
    sendMessage(q.sourceNode, message);
```

Figure 5.13: The pseudocode for in-network trie-based aggregation.

Caching to Improve Routing Latencies

The intermediate peer nodes in the query trie do not maintain state about the queries. As a result, every time an aggregation message is forwarded up the trie, the node has to perform a lookup, which involves multiple nodes in the overlay. To improve the performance of the system, each peer node maintains a small cache containing the most recently used prefixes, and the destination peer nodes corresponding to these prefixes.

Managing system dynamics

A new peer **joining** the system will take a part of another peer's load (i.e. its logical successor on the ring). A part of the recurrent queries stored locally at the peer will also be transferred to the joining peer, based on the longest prefix in their prefix lists. If the two peer identifiers share a prefix, and there are queries with that prefix, they will be copied to the new node and not moved, since there is not enough information to make a decision as to which node the query should be stored. A background algorithm then refines the queries until a prefix is obtained that is long enough to make a decision as to where the query belongs. An alternative solution is to do nothing and remove such queries when their *TTL* expires.

When a peer **leaves** the system it will send its data (including the recurrent queries and partial aggregates) to its successor. The lookup mechanism will identify the successor peer as the new intermediate node for ongoing partial aggregates.

Finally, peer **failures** affect the aggregation process in three ways: (1) the recurrent queries stored at the failed peer are lost, (2) the ongoing partial aggregates at the failed peer are lost, and (3) the queries that have been initiated at the failed peer (i.e. their source node) should be deleted from the system. The following paragraphs briefly discuss each situation. Note that the solutions proposed below have not been implemented in the current version of the system.

- Stored Queries: This problem occurs when the peer that failed was a trie leaf node for a recurrent query. In this case, the aggregates for that recurrent query will be incomplete. This problem is solved by replicating the recurrent query, along with the data, at the next *k* successors of this peer. The peers storing replicas are passive, they do not perform aggregates for that recurrent query. When the active peer fails, its successor becomes the active peer for that recurrent query.

- Ongoing Partial Aggregates: If the failed peer is an intermediate trie node performing partial aggregates for a query, these partial aggregates may be lost. This problem affects only the current aggregation process for that recurrent query. The identity of the failed node is known by the children peers (since they cache their parent node). The children peers monitor the health of the parent peer, and if the parent peer fails within a predefined time interval, a new parent is found and the partial aggregate is resubmitted. This is done by performing a lookup using the prefix of the parent trie node.
- Query Source: If the peer that submitted a recurrent query fails, that query has to be removed from the system. The failure of the query source node is detected by its children during the aggregation process. The children initiate a query invalidation process, similar to the query resolution process.

Load Balancing

The trie-based aggregation scheme may result in load imbalance. The peer nodes corresponding to trie nodes with short prefixes will be shared between multiple recurrent queries, and may experience heavier traffic and workloads. The solution is to distribute the query processing into the overlay such that the peer nodes near the root of the trie have a larger number of children than the ones at the bottom. This is based on the observation that, as the number of refinements increases, the sub-queries prefixes are longer, and the chances to have a large number of query trie nodes sharing the same prefix decrease. This way, the peers at the top of the query tries will be shared by fewer queries, reducing both the traffic and workload per node.

Building Geographical Awareness Into the Overlay

The SFC-based indexing scheme preserves data locality, which means that data stored at neighboring peers may come from the same region of the multidimensional space. Further, the replication scheme, in case of node failures, replicates data at successors of the peer node that stores it. It would be very efficient to form the overlay such that neighboring peers in the overlay are neighbors in the physical network as well. If all peers know their location (e.g. longitude and

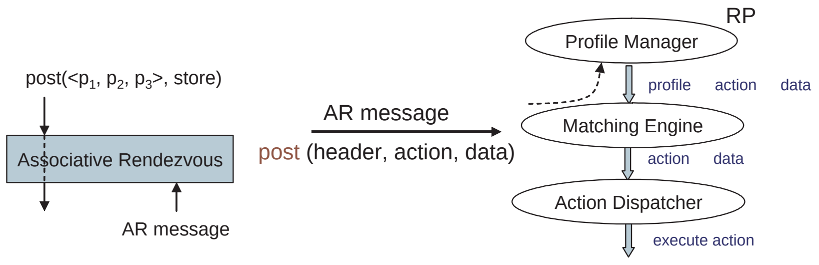


Figure 5.14: Profile manager and matching engine at a *rendezvous point*.

latitude), a SFC-based indexing scheme can be used to create node identifiers from the node's location descriptors, i.e., coordinates. As a result, geographical locality is preserved to some extent.

5.3.4 Associative Rendezvous Messaging Substrate

The AR messaging layer builds on top of content-based routing layer and provides the abstractions to enable decoupled interactions and reactive behaviors between peer elements using the unified interface: **post**(*profile, action, data*).

The ARMS layer implements the Associative Rendezvous interaction model. At each RP, ARMS consists of two components: the *profile manager* and the *matching engine*. The matching engine component is essentially responsible for matching profiles. An incoming message profile is matched against existing interest and/or data profiles depending on the desired reactive behavior. If the result of the match is positive, then the action field of the incoming message is executed first, followed by the evaluation of the action field for matched profiles. The profile manager manages locally stored profiles, and monitors message credentials and contexts to ensure that related constraints are satisfied. For example, a client cannot retrieve or delete data that it is not authorized to. The profile manager is also responsible for garbage collection. It maintains a local timer and purges interest and data profiles when their *TTL* fields have expired. Finally, the profile manager executes the action corresponding to a positive match.

5.3.5 Implementation Overview

Meteor builds on Chord (or our two-level extension to Chord). Chord, Squid, the ARMS layers, and the in-network aggregation service of the Meteor stack are currently implemented as event-driven JXTA services, so that each layer registers itself as a listener for specific messages, and gets notified when a corresponding event is raised. Project JXTA (http://www.jxta.org) is a general-purpose peer-to-peer framework that provides basic peer-to-peer messaging services. Since Meteor is designed as an overlay network of rendezvous peers, it is incrementally deployable. A joining RP uses the Chord overlay protocol and becomes responsible for an interval in the identifier space. In this way, the addition of a new rendezvous node is transparent to the end-hosts.

The overall operation of the Meteor overlay consists of two phases: bootstrap and running. During the bootstrap phase (or join phase) messages are exchanged between a joining RP and the rest of the group. During this phase, the RP attempts to discover an already existing RP in the system to build its routing table. The joining RP sends a discovery message to the group. If the message unanswered after a set duration (in the order of seconds), the RP assumes that it is the first in the system. If a RP responds to the message, the joining RP queries this bootstrapping RP according to the Chord join protocol and updates its routing tables to reflect the join.

The running phase consists of stabilization and user modes. In the stabilization mode, an RP responds to queries issued by other RPs in the system. The purpose of the stabilization mode is to ensure that routing tables are up to date, and to verify that other RPs in the system have not failed or left the system. In the user mode, each RP interacts at the Squid and ARMS layers.

5.4 Experimental Evaluation

A prototype of Meteor has been deployed on (1) a local-area network of 64 Intel Pentium-4 1.70GHz computers with 512MB RAM Linux2.4.20-8 (kernel version) and an 100 Mbps Ethernet interconnect, (2) the PlanetLab [93] wide area testbed, which is a global scale heterogeneous distributed environment composed of interconnected sites with various resources, and (3) the ORBIT [2] wireless testbed, which is composed of hundreds of wireless nodes, and is used to test wireless environments for pervasive applications. In these deployments, each peer node serves as a Rendezvous Point (RP) and executes an instance of the Meteor stack.

An experimental evaluation of the average performance of Meteor with aggregation service on up to hundreds of nodes using these deployments is presented below. Further, an evaluation of the scalability of the aggregation service on up to thousands of nodes using simulations is presented.

The system was populated with randomly generated data, resulting in a uniform distribution. The data items used were regular XML profiles, consisting of location attributes (e.g. RP's longitude and latitude), content descriptors (keywords) and a *TTL* field. A set of recurrent queries was stored into the system, typically at multiple RP nodes (e.g. the queries contained ranges, wildcards, partial keywords). Each recurrent query had associated a *TTL* and a field indicating the evaluation frequency. The queries were evaluated as they reached the destination nodes, and the results were aggregated and propagated towards the requesting RPs.

The metrics used in the evaluation included query dissemination time and query aggregation time. Query dissemination time is the time it takes for a query to reach all its destination nodes. Query aggregation time is the query dissemination time plus the time it takes to perform the aggregations and back propagate them.

Two sets of experiments were performed. The first one measured the performance of the aggregation service, considering a stable system, with no nodes joining, leaving or failing. The second experiment measured the robustness of the aggregation service in the presence of failures.

The experiments performed over different environments are based on a single Chord ring overlay unless explicitly stated. The results presented in the following section demonstrate that Meteor can effectively scale to large number of peers while maintaining acceptable execution time for messages.

5.4.1 Scalability of Aggregation Services

This experiment examines the scalability and efficiency of the aggregation service on ORBIT, PlanetLab and local LAN environments. Meteor has been deployed on up to 250 nodes on the ORBIT wireless testbed. The ORBIT large-scale radio grid emulator [2,97] consists of an array of 20x20 open-access programmable nodes each with multiple 802.11a,b,g or other (Bluetooth, Zigbee, GNU) radio cards. The overlay used in this deployment was constructed as follows.

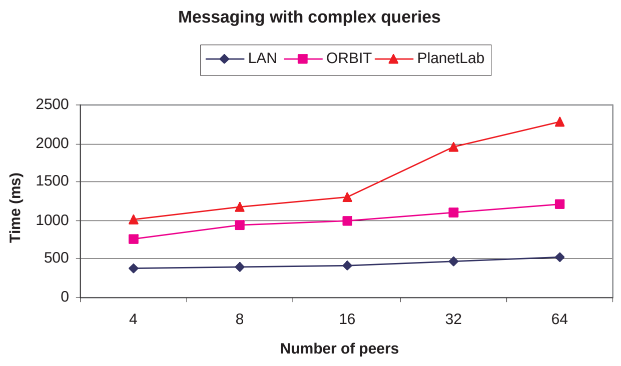


Figure 5.15: Scalability of aggregation service in term of overlay size for Orbit testbed, PlanetLab testbed, as well as LAN

The wireless nodes were divided into groups of up to 64 nodes. Each group was associated with an access point (AP). The APs were connected using a wired interconnect via Chord ring overlay. Nodes were allowed to randomly join or leave a group. Note that communication latencies varied significantly with time and location within a group.

Meteor has been deployed on over 60 nodes on the PlanetLab wide area testbed. Since PlanetLab nodes are distributed across the globe, communication latencies can vary significantly with time and node location [66]. In each of the experiments presented below, at least one node was selected from each continent, including Asia, Europe, Australia, and North America. Nodes randomly joined the Meteor system during bootstrap phase, resulting in a different physical construction of the ring overlay in each run. The experiments were conducted at different times of the day during a 4-week period. Once again, three sets of experiments were performed, one for each messages type. The average runtime for these experiments are plotted in Figure 5.15.

The aggregation queries used included ranges to specify the location of interest, (e.g., 15-37) and wildcards such as *temp**. The aggregation operator used was *COUNT*. The aggregation time for different network sizes on different deployment environments are plotted in Figure 5.15. As shown in the figure, the aggregation time scales well in terms of system size on these three different environments. Note that, the average aggregation time in the wireless environments is in between of the wide-area environments and more stable LAN environments.

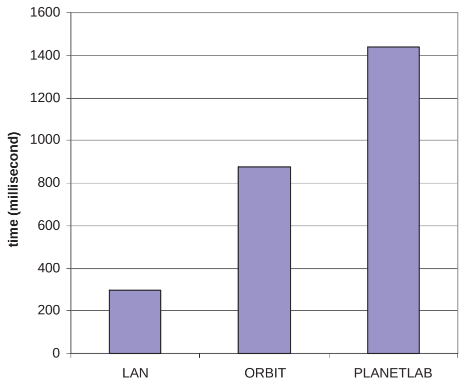


Figure 5.16: Average runtime for complex messages for LAN, ORBIT and PlanetLab

The aggregation time includes overheads such as routing table lookup and aggregate computation. The overall latency shows that our system can provide near real-time pervasive services for complex queries required by scientific applications.

Figure 5.16 plots the average runtime for complex messages for LAN, ORBIT and PlanetLab environments. The average runtime for LAN is about two to three times smaller than for ORBIT, and five times smaller than for PlanetLab, which is expected due to the wireless connectivity and higher wide-area latencies, respectively. To summarize the experiments, in case of LAN environments, the overall runtime is dominated by content routine overheads of the content routing layer, while for the PlanetLab testbed, the overall runtime dominated by latencies at the network layer, and for the ORBIT testbed, the overall runtime is mainly composed of both network latency and content routing layer. The overhead of the AR layer is almost the same in all cases. Further, while the runtime in all three cases does increase with system size, its rate of increase much slower indicating the scalability of Meteor.

Robustness

The next experiment explores the behavior of the aggregation service in the presence of RP failures. More specifically, the experiment measured the time needed to recover the ongoing aggregation operation when the RP performing it fails. This includes the time it takes to detect the failure (the children peers use a timeout to detect the parent's failure), the lookup time for the current logical parent (based on the trie prefix), and the time to re-send the partial aggregates to the new parent peer. Also, the local cache was updated with the new peer responsible for the trie prefix. Figure 5.17 shows that the time needed to recover partial aggregates is scalable in terms of system size.

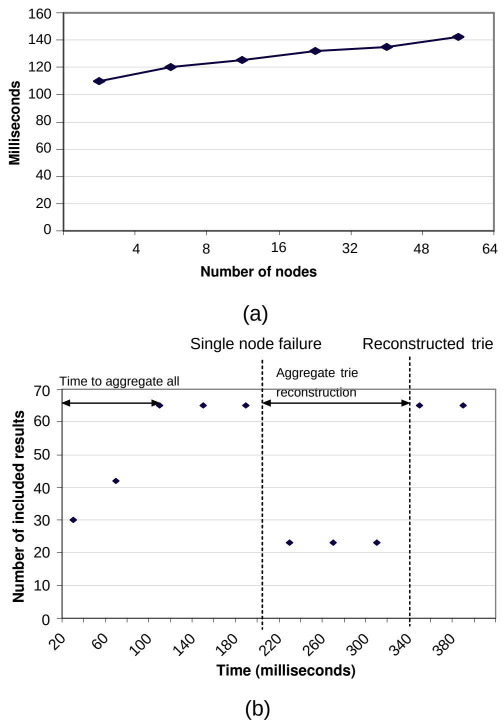


Figure 5.17: Robustness for single peer failure: (a) time to recover the ongoing aggregation, (b) number of aggregated results during peer node failure.

Figure 5.17 (b) shows how a peer node failure affects the aggregation process. In this experiment the same query is evaluated at regular intervals of time, over a period of 400 milliseconds. The aggregation results are propagated up the trie to the requestor. The size of the system is constant, and the quantity of data stored is also constant during this experiment. The expected result of the COUNT aggregation operation is 67. The first two aggregation operations result in a smaller COUNT value. This is because, even if the system is stable, not all peers will begin evaluating the query and aggregating the results at the same time. Also, the latencies between nodes vary. The next three aggregation operations are successful. As the picture shows, the system experiences a peer node failure at time 205. The on-going aggregates at this node are lost, and it takes about 100 milliseconds to detect the failure and correct the aggregation trie.

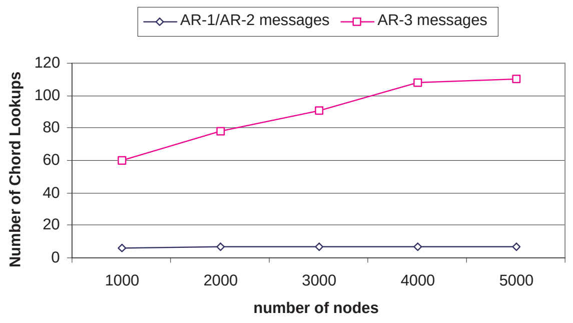


Figure 5.18: Simulation results for large scale system

5.4.2 Simulation Results

The performance of the Meteor infrastructure is also evaluated using a simulator. The simulator implements the Associative Rendezvous messaging, SFC-based mapping and the Chord-based overlay network for a network of up to about 5000 nodes. The simulator models wide-area network latencies between RPs using network delays statistics collected from the PlanetLab deployment. The resulting execution time of AR messages consists of overhead for AR messaging, Squid content-based routing and Chord overlay network delay including network delays. As the overlay network configuration and operations are based on Chord, its maintenance costs are of the same order as in Chord.

As shown in Figure 5.18, the average execution time for **AR-1** and **AR-2** messages is about 750 milliseconds for an overlay of 1000 *RPs* and becomes about 850 milliseconds for about 5000 *RPs*. This illustrates the scalability of Meteor infrastructure for system of over thousands of nodes. For **AR-3** and **AR-4** messages the runtime is about 3.5 seconds in a 1000 overlay network and becomes about 6 seconds for a network with 5000 *RPs*. The increase of runtime is mainly due to the underlying multi-hop network delays, which represents approximately 85% for **AR-1/AR-2** messages for thousands of RPs. For **AR-3** messages the network latency represents 88% of overall overhead. As the overlay network configuration and operations are based on Chord, its maintenance costs are of the same order as in Chord. From Figure 5.18, we can conclude that Meteor scales well to large systems of thousands of *RPs*.

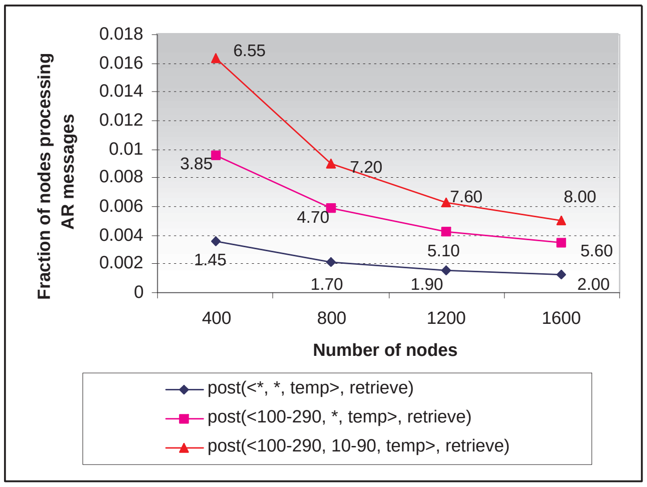


Figure 5.19: Simulation result of the aggregation service on larger systems. The number of nodes processing aggregates before normalized to system size is also shown on the curve.

Effect of specifying location attributes for AR aggregation

The simulator was also used to evaluate the operation of the aggregation service for systems with up to 1600 RP nodes. This simulation used three sets of AR messages with complex profiles:

- no specification of location, e.g., $< *, *, temp >$
- specifying location range in one dimension, e.g., $< 100 - 290, *, temp >$
- specifying location range in two dimensions, e.g., $< 100 - 290, 10 - 90, temp >$

The number of peer nodes that processed the aggregates was measured for each AR message. The fraction of peer nodes processing each type of aggregation message, function of system size, is plotted in Figure 5.19.

The figure shows that the number of nodes that process a message is a small fraction of the system size. Further, while the number of nodes processing a specific message increases with the system size, it increases at a slower rate than the system size, indicating the scalability of the system. Finally, the results demonstrates that as the specification of the profile become more specific, the fraction of nodes involved decreases, indicating the effectiveness of the Meteor routing mechanism.

Effectiveness of In-network Aggregation Service

This experiment measured the effectiveness of in-network aggregation when compared with the straightforward approach (e.g. the local aggregates are sent directly to the requestor, which performs the final aggregation). The experiment was conducted using a simulator with up to 2000 RPs. A set of aggregation queries (i.e. range queries) were used for the measurements.

For each query we measured the number of aggregation messages per node, with and without in-network aggregation. The results were aggregated and presented in Figure 5.20 (a). Figure 5.20 (b) shows the actual number of messages for one query, with in-network aggregation, evaluated in a system with 500 nodes.

As it can be seen in Figure 5.20 (b), by distributing the aggregation process in the network, each node handles a small number of messages. Also, the processing load is distributed. The number of messages per node grows slowly with the size of the system, demonstrating the scalability of in-network aggregation.

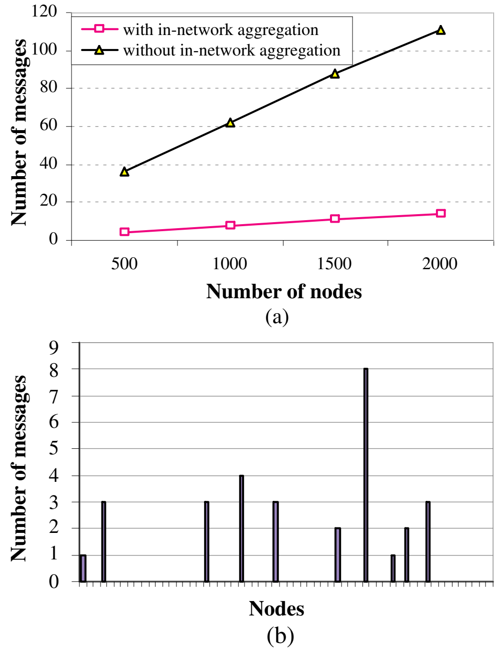


Figure 5.20: Effectiveness of in-network aggregation service: (a) average number of messages per peer with and without in-network aggregation; (b) actual number of messages per peer for a query aggregated in-network, for a 500 peers system.

5.5 Summary

As the scale, complexity, heterogeneity and dynamism of pervasive grid environments increase, interaction paradigms based on static names (addresses, identifiers) and on synchronous or strictly coupled interactions are quickly becoming insufficient. This has lead researchers to consider alternative paradigms that are decoupled and content-based. This chapter presented a content-based decentralized information aggregation service for sensor-based pervasive environments. The service provides a uniform query interface where aggregation queries are specified using content descriptors from a semantic space in the form of keywords, partial keywords, wildcards and ranges, and an action descriptor defining the aggregate operation. The service guarantees that all data that matches a query will be located and aggregated. Further, aggregation queries may be recurrent, i.e., they can be defined to execute repeatedly at predefined constant intervals.

The aggregation service was built on the Meteor content-based middleware infrastructure and extended the Associative Rendezvous (AR) model for content-based information discovery and decoupled interactions. Specifically, it extended the AR abstractions to enable content-based aggregation queries to be flexibly specified using keywords, partial keywords and ranges. Further, it used the Squid content-based routing infrastructure to construct aggregation tries so that query propagation routes can be used for back-propagating and aggregating matching data elements. The deployment and experimental evaluation of the aggregation service were also presented. The evaluations included simulations as well as experiments using deployments on local-area network at Rutgers and wide area PlanetLab testbed. The experiments demonstrated the scalability, effectiveness and robustness of the aggregation service in distributed and dynamic pervasive environments.

Chapter 6

Enabling End-to-end Sensor-driven Scientific and Engineering Applications

Technologies in surveillance and and sensing are becoming available in an increasing number of scientific and engineering applications. The deployment of sensors is offering unlimited possibilities to monitor and obtain a dynamic understanding of the different processes taking place at different spatial and temporal scales. In spite of the enormous potential benefits, the effective and efficient integration of this data into the management process presents significant challenges in data acquisition, assimilation and its integration in the computational models and the decision making process. The previous chapters presented a programming system for end-to-end sensor/actuator-based scientific and engineering applications, which provided semantically meaningful abstractions and runtime mechanisms for integrating sensor systems with computational models for scientific processes, and for in-network data processing such as aggregation, adaptive interpolation and assimilations. In this chapter, an end-to-end oil reservoir application that combines reservoir simulation models with sensors/actuators in an instrumented oilfield is used as a case study to demonstrate the operation of the programming system, as well as to experimentally demonstrate its effectiveness and performance.

6.1 Introduction

In many current scientific and engineering applications, modeling to predict system behavior is largely done using static historical data. This approach makes it impossible for such models to accurately predict temporal and spatial variations in the real-world. With advances in sensor technology, it now becomes possible to feed in near real time measurements data from diverse, complex, distributed sensor networks, enabling more accurate modeling, prediction and control. The effective and efficient integration of this data into the scientific applications, such as the management and control of subsurface geosystems, presents significant challenges in data acquisition, assimilation and its integration in the computational models and the decision making process.

The overall goal of the research effort is to investigate sensor system middleware and programming support that will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing as they stream data from the physical environment to the computational world [52]. Further, application should be able to interact with the sensor system to control sensing and data processing behaviors. The programming systems enables sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it supports programming models and systems for developing in-network data processing mechanisms. The former supports complex querying of the sensor system, while the latter enables development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilations, both via semantically meaningful abstractions. The research is driven by the management and control of subsurface geosystems, such as managing subsurface contaminants at the Ruby Gulch waste repository [92] and management and optimization of oil reservoirs [61]. Crosscutting requirements of these applications include multi-scale, multi-resolution data access, data quality and uncertainty estimation, and predictable temporal response to varying application characteristics.

The focus of this chapter is on the end-to-end abstractions provided by the programming system, and on how they can be used to enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems in a semantically meaningful way. Specifically, this chapter describes the usage and effectiveness of the *GridMap/iZone* abstractions using an *Instrumented Oilfield* application as a case study. A prototype programming system has been implemented. An experimental evaluation using this prototype is also presented.

The rest of the chapter is organized as follows. Section 6.2 demonstrates how to use the *GridMap/iZone* abstractions to implement such an end-to-end oil reservoir application that

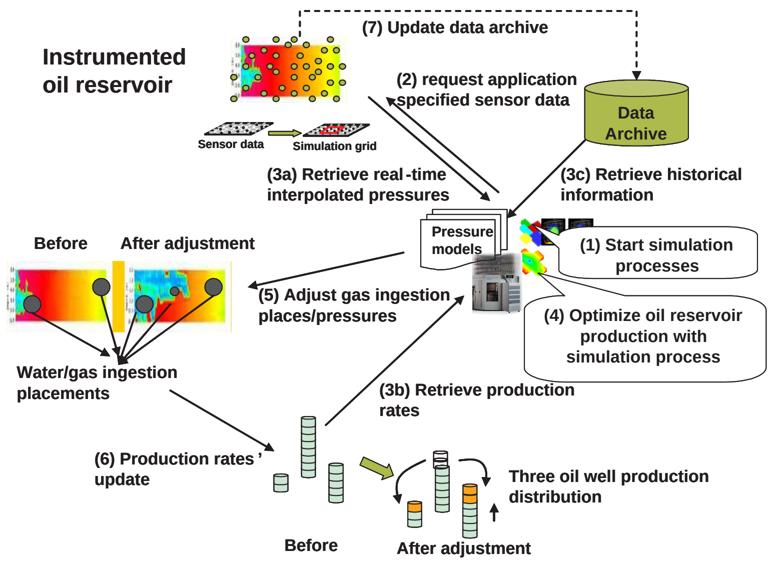

Instrumented oil reservoir
(7) Update data archive
(2) request application specified sensor data
Data Archive
Sensor data
Simulation grid
(3a) Retrieve real-time interpolated pressures
(3c) Retrieve historical information
Pressure models
Before
After adjustment
(1) Start simulation processes
(5) Adjust gas ingestion places/pressures
(4) Optimize oil reservoir production with simulation process
Water/gas ingestion placements
(3b) Retrieve production rates
(6) Production rates' update
Three oil well production distribution
Before
After adjustment

Figure 6.1: Overview of an end-to-end oil reservoir application.

combines reservoir simulation models with sensors/actuators in an instrumented oilfield. Section 6.3 presents an experimental evaluation of the programming system. Section 6.4 summarizes the chapter.

6.2 An End-to-end Oil Reservoir Application Using *GridMap/iZone* Abstractions

Scientific applications often require measurements at pre-defied grid points, which are often different from the locations of the raw data provided directly by the sensor network. As a result, the sensor-driven scientific and engineering applications require a virtual layer, where the logical representation of the state of the environment provided to the applications may be different from the physical representation of raw measurements from sensor network. The abstractions described in the previous chapters (see Figure 3.2 in Chapter 3) enable applications to specify such a virtual layer and the models (e.g. regression models, interpolation functions, etc.) that should be used to estimate data on the virtual layer from sensor readings, as well to develop in-network implementations of the data estimation mechanisms.

In this section, we demonstrate how the *GridMap/iZone* abstractions can be used to implement an end-to-end oil reservoir application that combines reservoir simulation models with sensors/actuators in an instrumented oilfield.

Subsurface behavioral surveillance and sensing is becoming available in an increasing number of environmental and energy reservoir applications. The deployment of sensors is offering unlimited possibilities to monitor and obtain a dynamic understanding of the different processes taking place at different spatial and temporal scales. Understanding and controlling these processes will certainly allow a better quantification of uncertainties and an up-to-date assessment of resources in these subsurface applications. A dynamic data driven management [61, 92] process can complete the feedback loop between measured data and computational models to provide more efficient, cost-effective and environmentally safer production of oil reservoirs as well as management and optimization of other subsurface geosystems, which can result in enormous strategic and economic benefits. Consequently, the oil industry is increasingly using instrumentation and online monitoring to increase productivity, reduce accidents and enable decision making during planning and operation.

However, in spite of the enormous potential benefits, the effective and efficient integration of this data into the reservoir management process presents significant challenges in data acquisition, assimilation and its integration in the computational models and the decision making process. Reservoir data is disparate, sparse and subject to different subjective interpretations, and the derivation of knowledge from reservoir data involves merging different sources and scales of data into a single consistent characterization unit, their geological and engineering interpretations, and the concatenated utilization of different simulation models.

The programming abstractions and systems software solutions can help address these challenges and enable end-to-end management processes of an instrumented oil field consisting of detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface. The overall goal is to ensure near optimal operation of the reservoir in terms of profitability, safety and/or environmental impact.

For example, the productivity of an oilfield can be predicted based on the current physical characteristics of oilfield. The productivity index and the reservoir pressure determine the oil production rate from a well for a given time period, and the latter is nonlinearly related to the cumulative oil produced. The well is usually capped when the GOR (gas to oil ratio) exceeds a certain threshold limit or when the pressure of the reservoir is lower than a minimum

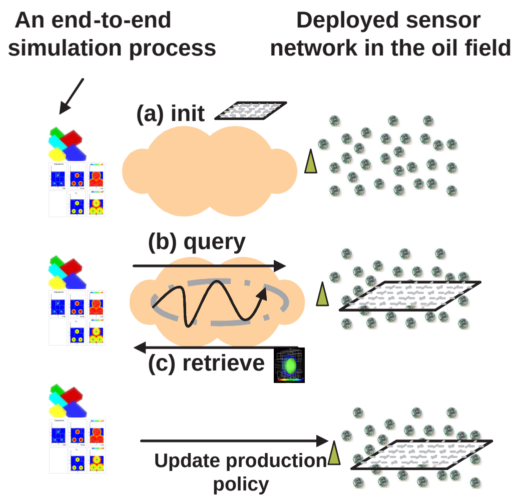

An end-to-end simulation process
Deployed sensor network in the oil field
(a) init
(b) query
(c) retrieve
Update production policy

Figure 6.2: Using *GridMap/iZone* abstractions to program the sensor-driven application.

pressure. In this example, the pressures and concentrations are modeled as dynamic fluids and are measured and simulated in the region of interest for well management. An overview of this application scenario is shown in Figure 6.1. The figure illustrates the steps involved in constructing a closed control loop for optimal reservoir management, including computational processes for issuing queries to instrumented oil reservoir, retrieving the relevant data and integrating it with the simulation processes, and making appropriate decisions for updating oil production policy.

The evolution of pressure and concentration in the oil field during production are simulated using comprehensive mathematical models of the subsurface. As the simulations evolve, these models periodically update pressure and concentration distributions in particular regions (e.g., regions requiring adaptive refinement due to high errors) from the oil field. Sensors deployed in the oil field monitor and retrieve current pressures and/or concentrations in regions of interest. The results of the simulations are then used by the production optimization process to generate optimal configuration (i.e. production rate, gas, water pressures, etc.). The realization of these steps using the *GridMap/iZone* abstractions are briefly described below:

The application first uses the *GridMap* init operator to construct a virtual sensor grid overlayed on the oilfield, to match the grid used by the simulations models. For example, a *GridMap* instance constructed using the specification $<x_{lb}:\delta_x:x_{ub},\ y_{lb}:\delta_y:y_{ub}>$ represents a two dimensional rectangular virtual grid with its left bottom corner at (x_{lb}, y_{lb}) and top right corner at

(x_{ub}, y_{ub}), and with a spacing of δ_x and δ_y along the x and y dimension respectively. The application can also specify the interpolation method, for example, “closest neighbor”, “IDW” or “Kriging”, to be used to compute data values at the virtual grid points using sensor values.

The application can now “query” the sensor system using regions of the virtual grid. The application query specifies the data types of interest (e.g., pressure, concentration), error thresholds, etc., using the syntax described in Section 3.5. The queries are forwarded by the runtime system to appropriate sensors nodes as shown in Figure 6.2 (b). For each virtual grid point, *iZones* are constructed by discovering all relevant sensors. The data retrieved from these sensors is then interpolated onto the required virtual grid points using the specified interpolation method and the in-networks mechanisms provide by *iZone*. The resulting data values on the virtual grid are then returned to the application.

The application can now continue the simulations using the updated data, and along with current production rate, historical data, etc, to, for example, predict changes in oil reservoir, evaluate different configurations, and optimize desired objective functions such as maximizing production rate and/or minimizing cost (e.g. gas ingestion cost). The overall data flow of the end-to-end oil reservoir management/optimization process is illustrated in Figure 6.3.

6.3 Experimental Evaluation

The parameter estimation and oil reservoir optimization scenarios in an instrumented oilfield (described in Section 6.2) is implemented using the prototype of the *GridMap/iZone* programming system described above and is used for the experiments presented in this section. The objectives of the experiments are not only to demonstrate the ability of *GridMap/iZone* to support the integration of computational processes with run-time in-network processing of sensor data, but also to evaluate the effectiveness and efficiency of the prototype implementation for realistic scenarios. The scenarios in the experiments are driven by a real-world sensor dataset obtained from an instrumented oil field with 2000 sensors, and consisting of pressure measurements sampled 96 time per day. A two-tiered sensor overlay with 40 clusters was emulated on 40 nodes of the Orbit wireless testbed so that each cluster ran on a single node of the testbed. The sensors are assumed to be randomly distributed in the sensors field. The computational

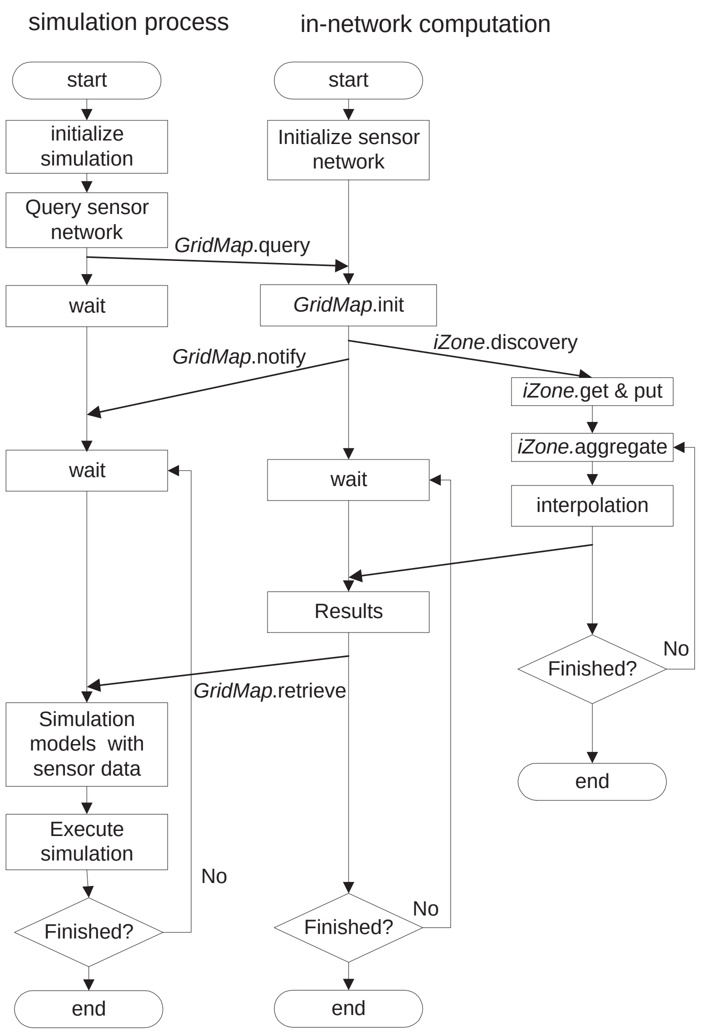


Figure 6.3: Overall dataflow of the end-to-end oil reservoir management/optimization process.

processes ran on compute cluster located at a different campus at Rutgers.

The end-to-end cost of an interaction between a computational process and the sensor system consists of 5 parts as illustrated in Figure 6.4: (1) the cost of issuing the *GridMap* query by a computational process, and routing it through the network to the sensor network gateway; (2) the cost of forwarding the query from the gateway to all relevant sensors associated with the *GridMap*; (3) cost of in-sensor-network computations (e.g., interpolation) associated with a query; (4) the cost of aggregating the data and forwarding it to the gateway; and (5) the cost returning the results back to the computational process. The experimental evaluation in this chapter focuses on the in-sensor-network costs, i.e., (2), (3) and (4). These costs are measured in terms of the number of messages, the number of hops per message and the volume of data

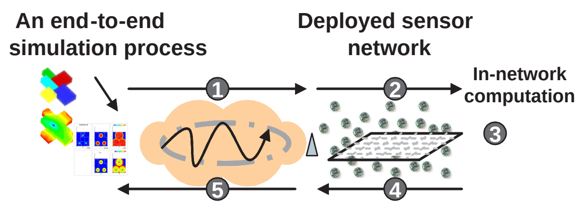


Figure 6.4: The components of performance evaluation

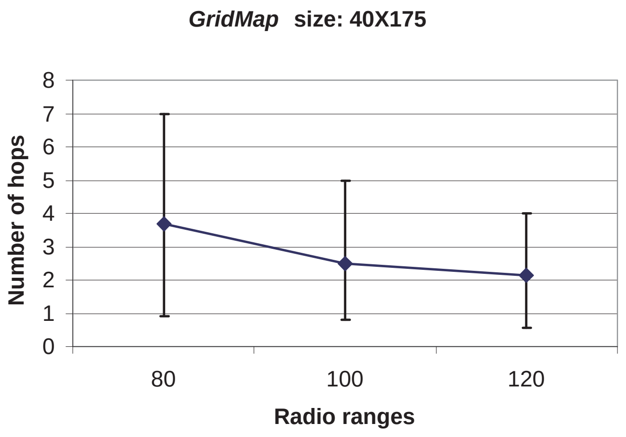
transferred, and the impact of overall size of the *GridMap*. The tradeoffs between the accuracy of data estimation and associated costs are also evaluated.

Communication Costs

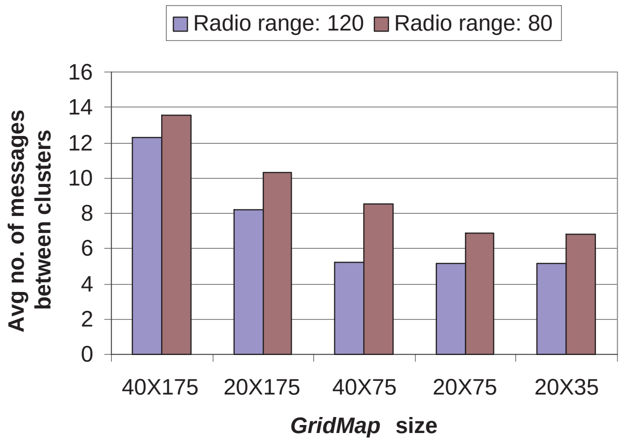
This experiment measures the communication costs, in terms of average number of hops, of querying all the sensors associated with a *GridMap*. The first set of experiments keep the size of the *GridMap* fixed at 40ft x 175ft, and varies the radio range of the sensors. The results are plotted in Figure 6.5(a). As expected, as the radio range increases, the average number of hops decrease. The next set of experiments measured the impact of increasing the size of the *GridMap* on the average number of messages between clusters in the sensor network for two different radio ranges. The results are plotted in Figure 6.5(b). As the size of the *GridMap* increases, so does the average number of messages required per update. For example, when the size of *GridMap* is doubled,e.g., from 20ft x 175ft to 40ft x 175ft in the experiment, the average number of messages increase by about 1.5 times. Additionally, as the size of the *GridMap* becomes smaller (e.g. 40ft x 75ft, 20ft x 75ft and 20ft x 35ft for a radio range of 120ft), the number of messages does not change much, partially because almost all of the involved clusters of the given *GridMap* are within a single hop of each other.

Tradeoff between accuracy and communication costs

In this experiment, we examine the tradeoff between accuracy and communication costs for cases where the data processing (i.e., interpolation) is done within the sensors system and external to the sensor system (for example, at a remote server). The latter case represents the conventional approach where raw data is collected from the sensor network and transferred to a server where required processing is performed in an offline manner. In this case the costs



(a)



(b)

Figure 6.5: Communication costs of querying all the sensors associated with a *GridMap* as a function of the size of the *GridMap* and the radio range of the sensors.

measured in terms of the data volume that is transferred to the gateway. The accuracy of interpolation is determined in terms of the interpolation error and depends on the interpolation mechanism used. To ensure a fair comparison, the same data processing is used in both cases.

The results are plotted in Figure 6.6. As seen in the plots, increasing the required accuracy increase the volume of communication. For in-network interpolation, this is because more data is required for each *iZone*. Furthermore, as the size of the *GridMap* increases, the volume of communication also increases. Note that the communication cost does not grow linearly with the number of grid points over the same *GridMap*. This is partially because part of the data is shared between neighboring *iZones*, therefore eliminating the need for some of the

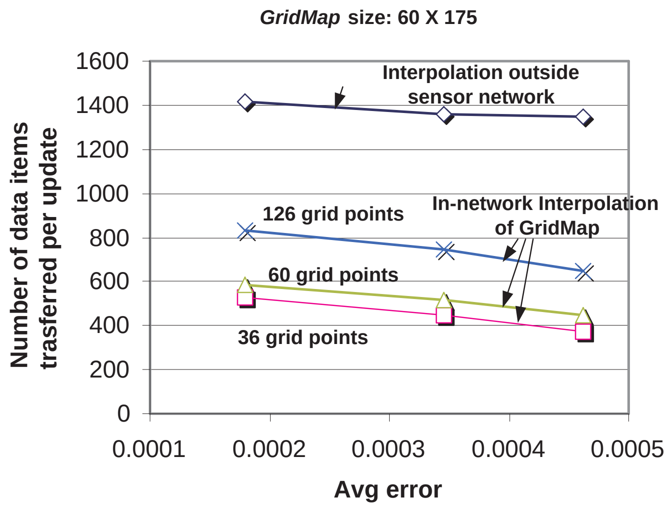


Figure 6.6: Tradeoff between accuracy and communication costs for different sizes of the *GridMap*.

communications when computing the grid points.

For the case where the interpolation is performed outside the sensor network, the data volume does not change significantly with increasing accuracy since only a small additional amount of data is needed.

A key observation is that for the same level of accuracy, the communication volume is significantly large when the interpolation is performed outside the sensor network, which also implies increased bandwidth consumption, latencies as well as energy costs. This demonstrate a key benefit of in-network data processing, specially in the case of application requiring near real-time analysis of and reaction to sensed data.

6.4 Summary

This research presented a programming system for end-to-end sensor/actuator-based scientific and engineering applications, which provided semantically meaningful abstractions and runtime mechanisms for integrating sensor systems with computational models for scientific processes, and for in-network data processing such as aggregation, adaptive interpolation and assimilations. Specifically, the programming system provides the end-to-end *GridMap* abstraction that enables computational applications to access and integrate sensor data into their models, and the in-network *iZone* abstraction to enable the development of scalable in-network data processing mechanisms. The underlying middleware provides an in-network data processing engine, which supports efficient data dissemination, aggregation and collaboration in dynamics, resource-constraint heterogeneous sensor networks. In this chapter, an end-to-end oil reservoir application that combines reservoir simulation models with sensors/actuators in an instrumented oilfield was used as a case study to demonstrate the operation of the programming system, as well as to experimentally demonstrate its effectiveness and performance.

Chapter 7

Enabling Autonomic Power-Aware Management of Instrumented Data Centers

Sensor networks support flexible, non-intrusive and fine-grained data collection and processing and can enable online monitoring of data center operating conditions as well as autonomic data center management. This chapter describes the architecture and implementation of an autonomic power aware data center management framework, which is based on the integration of embedded sensors with computational models and workload schedulers to improve data center performance in terms of energy consumption and throughput. Specifically, workload schedulers use online information about data center operating conditions obtained from the sensors to generate appropriate management policies. Furthermore, local processing within the sensor network is used to enable timely responses to changes in operating conditions and determine job migration strategies. Experimental results demonstrate that the framework achieves near optimal management, and in-network analysis enables timely response while reducing overheads.

7.1 Introduction

Computing and communications are an integral part of society's IT infrastructure, affecting every aspect of life, including services related to health, banking, commerce, defense, education and entertainment. The increasing demands for computing and storage, and as a result, the growing scales of enterprise computing environments and data centers have made issues related to power consumption, heat generation and cooling requirements of critical concern – both in terms of the growing operating costs (power and cooling) as well as their environmental and societal impacts. In fact, the Environmental Protection Agency (EPA) recently reported that the energy used by datacenters by the year 2011 is estimated to cost \$7.4 B (15 power plants, 15 Gwatts/hour peak) [5]. Furthermore, power and cooling rates are increasing by an alarming 8 fold every year and are becoming the dominant part of IT budgets. Clearly addressing this problem is an important and immediate problem for enterprise data centers.

This chapter addresses the autonomic management of instrumented data centers, which is based on the integration of online data from sensor networks with the computational processes that model physical phenomena in the data center as illustrated in Figure 7.1, with the overall goal of optimizing data center performance in terms of energy consumption and throughput. Specifically, this chapter focuses on the architecture and programming/runtime system required to facilitate the interactions between computational models and the sensor system and support such an integrated management approach. The key challenges and requirements addressed are as follows [54].

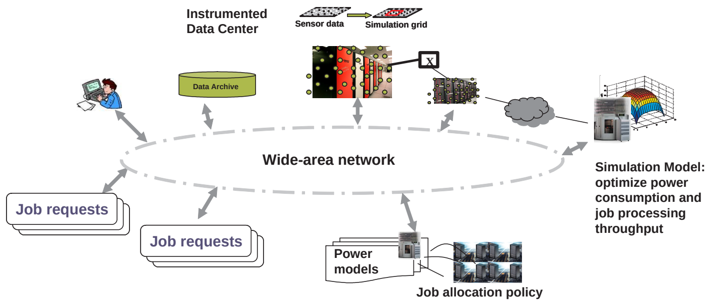


Figure 7.1: A sensor-driven data center management scenario.

Real-time end-to-end sensing and control: Traditionally, the configuration of a data center is based on a snapshot of thermal conditions, which is often derived from spatially dense measurements obtained using an sensing system. Such a representation presents an accurate snapshot of the thermal conditions in a data center only at the time of measurement. Over time, however, the configuration of a data center's computing devices, including networking and storage devices, as well as as the air flow conditions and associated cooling system, are all subject to changes. A more dynamic measurement and modeling approach is thus needed to provide real-time status data, and possibly to enable a predictive evaluation of dynamic scenarios. This can be achieved by deploying real-time sensors, which deliver measured data either at regular intervals or upon occurrence of predefined events. Thus, by combining data from computational models, for example, air recirculation and/or heat distribution models, with real-time sensor data, an autonomic data center management system can be constructed.

Timely response to dynamics using in-network processing: The computational models, for example, those used to predict the heat/temperature distribution in the data center, can often have long run-times due to the complexity of the model themselves as well as the large volume of the data involved. By combining local processing within the sensor network with these longer time scale computational processes, a more timely response to critical events can be achieved.

Enhanced efficiency and finer granularity of control: The deployment of sensors enables a finer granularity of control. For example, sensors can monitor temperature and humidity, which are inversely correlated, and can watch for "hotspots" to improve the overall efficiency of the cooling system. Sensors can also be used to identify which systems are overheating and need to be powered down, or which servers are being overcooled and wasting energy.

To address above challenges and requirements, we propose a two-level framework to support the autonomic management and control of instrumented data centers. Specifically, the overall goal of the framework is to enable the effective integration of densely deployed sensors with optimization and computational processes to improve power consumption, efficiency of the cooling system and job throughput. Sensor networks monitor temperature, humidity, airflow, etc., in real time, provide non-intrusive and fine-grained data collection, and enable real-time processing. The sensor data is integrated with computational processes and middleware services such as job schedulers, to take phenomena such as heat distribution and air flow into account. For example, workload schedulers use online information about data center operating conditions obtained from the sensors to generate appropriate scheduling strategies. Furthermore, local processing within the sensor network is used to enable timely responses to changes in operating conditions and determine job migration strategies. Experimental results demonstrate that the framework achieves near optimal management, and in-network analysis enables timely response while reducing overheads.

Note that the objective of this chapter is not to present new data center management policies or mechanisms, but rather to demonstrate how such policies and mechanism can be implemented with an integrated end-to-end management process using the *GridMap/iZone* programming system, as well as highlight the advantages of such an integrated process. The programming system provides end-to-end abstractions and enables scientific/engineering applications to discover, query, interact with, and control instrumented physical systems in a semantically meaningful way.

The rest of the chapter is organized as follows. Section 7.2 describes the architecture of the autonomic data center management framework. Section 7.3 presents the use of the *Gridmap/iZone* programming system for integrating sensor systems with computational processes. An experimental evaluation is presented 7.4. Section 7.5 discusses related work. Section 7.6 concludes this chapter.

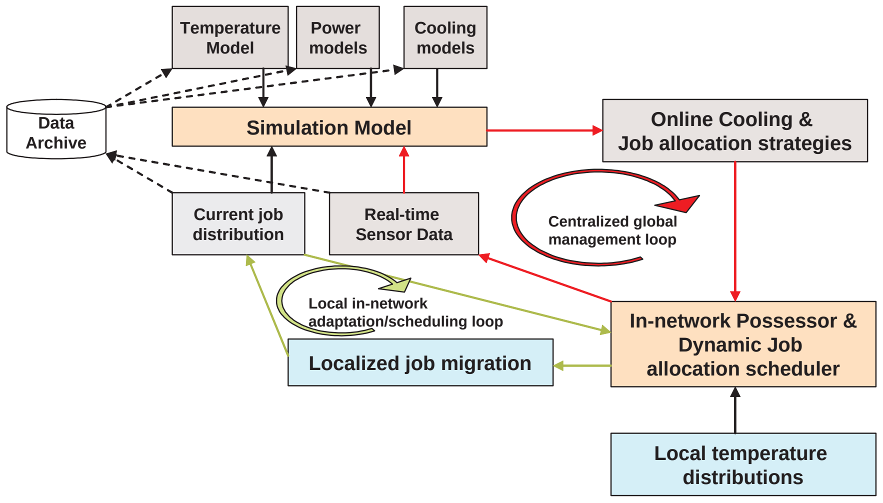

Temperature Model
Power models
Cooling models
Data Archive
Simulation Model
Online Cooling & Job allocation strategies
Current job distribution
Real-time Sensor Data
Centralized global management loop
Local in-network adaptation/scheduling loop
In-network Possessor & Dynamic Job allocation scheduler
Localized job migration
Local temperature distributions

Figure 7.2: A two-level autonomic data center management system.

7.2 Architecture of a Two-level Autonomic Data Center Management System

Recent trends toward dense blade computing, server virtualization as well as unbalanced load distributions contribute to increased rack power densities and introduce dynamic hot spots in the data center, and as a result, dense online monitoring systems are necessary to avoid potential consequences. For example, the collected data can facilitate online troubleshooting (e.g., hotspot alarms, uneven load distributions) and trigger preventive maintenance. In the data center management scenario addressed in this research, sensors monitor data center operating conditions such as temperature, humidity and airflow in real time, provide non-intrusive and fine-grained data collection, and enable online processing. The data collected from sensors and the derived data computed within the sensor network are then transported to compute servers where this data is integrated with computational processes and workload schedulers, which use this data to take issues such as heat distribution, air flow and cooling into account while making management decisions.

In this section, we present an autonomic data center management framework that supports the above scenario at two levels as illustrated in Figure 7.2. Reactive online management is supported through local processing and decision making within the sensor network. Proactive management is achieved using computational processes that use the data from the sensors and model the physical phenomena in the data center, evaluate different managements strategies and define longer term management policies. In the rest of this section, we describe the main components of the framework as well as its operation in more detail.

Sensor Data Collection and In-network Processing: The sensor network monitors and collects data about the operating conditions. Data can be directly collected from the sensor nodes or computed using in-network computations, e.g., interpolation. The collected sensor data is transported to the compute servers and used by the computational processes that, for example, model the thermodynamic phenomenon in the data center, or make mapping and scheduling decisions. Sensor data can also be locally processed within the sensor network to detect and possibly respond to events of interest, e.g., a hotspot.

Simulation and Computational Processes: The simulation models and computational processes often need global information about the data center in order to optimize its operations and make management and control decisions. The required information includes the data center operating conditions monitored by sensors, for example, to provide an initial temperature distribution to a thermodynamics model [10, 84, 111], or to provide inputs to learning models and/or prediction processes [84]. Models are also useful for correlating current operational data (e.g., temperature distributions) with historical data for data center under different job allocation policies.

Policy Selection and Workload Allocation: The results of the data analysis and computational processes are used to select appropriate data center operation policies as well as mapping and scheduling strategies, so as to optimize the overall operation of the data center in terms of energy consumptions, costs, throughputs, etc. For example, polices proposed in literature include "coolest inlet temperatures with lower number of localized hotspots" [84], "minimal power consumption at machines", "uniform outlet temperatures" and "minimal heat recirculation" [111]. Note, the focus of this research is not defining new polices but using online sensor data to select appropriate policies as well as mapping and scheduling strategies.

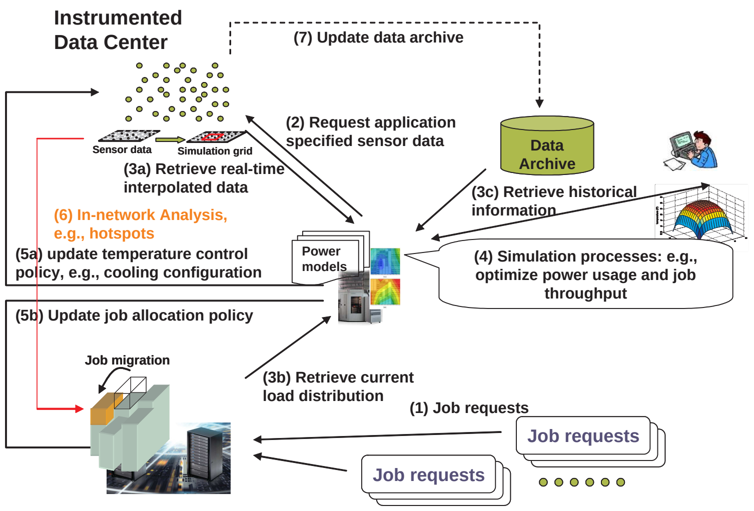

Instrumented Data Center
(7) Update data archive
(2) Request application specified sensor data
Data Archive
Sensor data
Simulation grid
(3a) Retrieve real-time interpolated data
(3c) Retrieve historical information
(6) In-network Analysis, e.g., hotspots
(5a) update temperature control policy, e.g., cooling configuration
Power models
(4) Simulation processes: e.g., optimize power usage and job throughput
(5b) Update job allocation policy
Job migration
(3b) Retrieve current load distribution
(1) Job requests
Job requests
Job requests

Figure 7.3: Enabling autonomic management of instrumented data centers using the *GridMap/iZone* programming system.

7.2.1 Two level Autonomic Data Center Management

The above architecture is used to enable autonomic data center management at two levels. At the local level, sensor data is analyzed within the sensor-network to identify and enable a timely response to policy violations or undesirable changes in the data center's operating conditions. For example, local sensor data may be used to detect a developing hot spot or a violation of current energy consumption policies, and as a result, adjust the workload allocation by migrating jobs. Note that the objective of management at this level is to make local adaptation to ensure that current policies are not violated. More global adaptation as well as policy changes are made in the more expensive (and less frequent) global management loop.

The global management loop operates at the data center level. It uses data about the entire data center and the applications, along with historical data and simulation models to generate longer-term policies and allocation strategies, for example, based on load distributions, temperatures, air flow and recirculation, response time, throughput, etc. The overall process is illustrated in Figure 7.2.

A key advantage of such a two-level approach is that the frequency of expensive data collection and global decision making can be reduced using local management. Furthermore, the local management can also react more quickly to undesirable change and policy violations. These advantages are demonstrated in the Section 7.4.

7.3 Enabling Instrumented Data Center Management with *GridMap/iZone* Programming System

As described above, the autonomic sensor-driven data center management strategy combines sensor systems, simulation processes, and online schedulers to improve energy efficiency and workload performance. This requires a programming system to integrate the sensor systems and models, as well as in network support to enable applications to discover, query, interact with, and control the instrumented physical system using semantically meaningful abstractions. The *GridMap/iZone* [51, 52] programming system addresses these requirements and includes abstractions and runtime mechanisms for integrating sensor systems with applications processes, as well as for in-network data processing such as aggregation, adaptive interpolation, event detections, and assimilation.

The *GridMap/iZone* programming system enables the two level instrumented data center management outlined above as follows. First, it provides programming abstractions for integrating sensor systems with computational models for application processes and with other application components in an end-to-end manner. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms. Specifically, the temporal and spatial correlation of sensor measurements are leveraged to tradeoff between the complexity of coordination among sensors and the savings that result from having fewer sensors for in-network processing, while maintaining the data quality required by the applications.

7.3.1 Using *GridMap/iZone* Programming System for Data Center Management

The use of the *GridMap/iZone* programming system for data center management is illustrated using a autonomic power aware instrumented data center scenario where the heat distribution model queries temperature sensors using the *GridMap* operator as illustrated in Figure 7.3. The query is routed to appropriate regions of the sensor network and the interpolated values at each grid point of the *GridMap* are computed using the *iZone* abstraction and its in-network mechanisms [51], either periodically or on-demand. Current workloads are also retrieved. This data is used by the simulation processes to correlate CPU utilization and power consumption models, and to generate energy consumption and workload allocation policies in order to optimize data center usage. This online monitoring of operating conditions together with simulation and computational processes adapt to changes in the data center environment to complete the end-to-end management and control loop.

To illustrate the use of the programming system, consider a scenario where in-network processes compute operating parameters such as local temperatures or differences of the temperatures within a neighborhood specified by the application and/or management process. When hotspots are detected, an event is triggered, for example, to locate a resource with the lowest temperature for job re-allocation. As illustrated in Figure 7.3, a part of the workloads (e.g., jobs at the hotspot) may be migrated to keep the operating conditions within the desired constraints for a longer time without requiring a complete re-allocation. Note that the region of interest that is monitored as well as the density of sensing can be changed at runtime to adapt to the dynamics of the data center operating conditions using the abstractions provided by *GridMap/iZone*.

7.3.2 Programming In-network Policies

This section discusses how different in-network polices can be implemented using in-network mechanisms and the *GridMap/iZone* programming system. As discussed earlier, the initial job allocation policies (e.g., a baseline uniform job allocation policy) are generated by simulation processes. When this policy is generated, it is also communicated to the sensor system and triggers reactive behaviors corresponding to the policy. For example, a simple behavior could

```
IN-NETWORK ANALYSIS:
query(GridMap, radius, FindMinimals (n))
1 GridMap = [<100:10:200, 50:5,100>]
2 For each sensor/cluster
3    compute associated grid point
4       get(data, radius)
5       interpolate at the grid point
6    receive the n minimal grid point
        values from neighbors
7    compute the n new minimal values
8    send(loc_1,loc_2,...,loc_n)
```

Figure 7.4: Example of in-network analysis of sorting n minimal grid point values at each node using *GridMap/iZone*.

be that if the average temperature measured is greater than the redline value, migrate part of job from that computer to other computing nodes where the temperature and/or the utilization rate is less than a threshold. The sensor network continues monitoring temperatures, and when machine temperature is higher than the redline value, an in-network job management/control action is activated. Other allocation policies, such as uniform outlet profiling, maximize minimal inlet temperature, can be selected and the corresponding in-network policies and algorithms can be implemented using the *GridMap/iZone* programming system. We discuss two examples of such in-network policies using *GridMap/iZone* below.

Move to machine with lowest inlet temperature: This is one of the most straightforward in-network policies. Under this policy, when the inlet temperature is greater than the redline temperature (e.g., external machine temperature is greater than 25 ^{o}C), the tasks on the machine need to move to other machines. One of the intuitive algorithms is to move tasks to machines with the lowest inlet temperature. In this case, the continuous sensor monitoring can be used to *detect the lowest temperature*. With this in-network policy, the in-network processing task is to find the lowest inlet temperatures from time to time. An in-network sorting algorithm is used to maintain a list of a few (usually 1 - 4) machines with the lowest inlet temperatures from time to time, as shown in Figure 7.4. The task is then migrated to a machine from this list. As a result, inlet temperature is maintained below the redline threshold in the data center without requiring expensive global scheduling across the whole data center.

Move to machine generating lowest recirculation: In unbalanced heat recirculation environments, minimizing maximal inlet temperature [111] can be used as centralized scheduling

```
IN-NETWORK ANALYSIS:
1 - 5 same as Figure 4
find lowest average and variance
    around neighbors: avgvar(n)
6    receive(res_k, loc_k) from neighbor k
7    compute avgvar(n neigb grid points)
        res = a x avg + b x var
8       (res, i) = min(res_k)
9    send(res_i, loc_i)
```

Figure 7.5: Example of an in-network algorithm for computing average and variance in the neighborhood of each grid point.

policy. That is, tasks are allocated to nodes generating lowest recirculation. The corresponding in-network algorithm can be designed so as to find the region that has a low inlet temperature and also has a neighborhood with a relatively low inlet temperature, i.e., for which low recirculation is indicated. The corresponding in-network policy is listed in Figure 7.5, and computes the weighted average and variance in the neighborhood of each grid point of the *GridMap* from time to time, and the region with the lowest average and variance is selected for re-assigning tasks at runtime.

Other in-network policies can be developed and implemented with the *GridMap/iZone* programming abstractions to adapt to the change of the operating conditions.

7.4 Simulation and Results

The effectiveness of in-network data processing for the autonomic management of instrumented data centers is evaluated using a simulated instrumented data center environment where cooling and energy consumptions models presented in [10] are used for simulating the operation of the data center with two rows of racks arranged in a physical cold aisle and hot aisle layout, and synthetic sensor data is obtained using the heat distribution equations [84, 111, 122]. Cold air is assumed to be supplied by a central air conditioner. The synthetic sensor data is integrated with simulation processes, data archives and a user subsystem through gateway nodes. Queries issued by the computational process are routed to appropriate sensor nodes via the gateway and cluster heads of the sensor network, and data values are aggregated and interpolated as they are routed back. The rest of this section focuses on an evaluation of the effectiveness and benefits of the proposed management approach, and especially the use of local in-network adaptations, using end-to-end application scenarios.

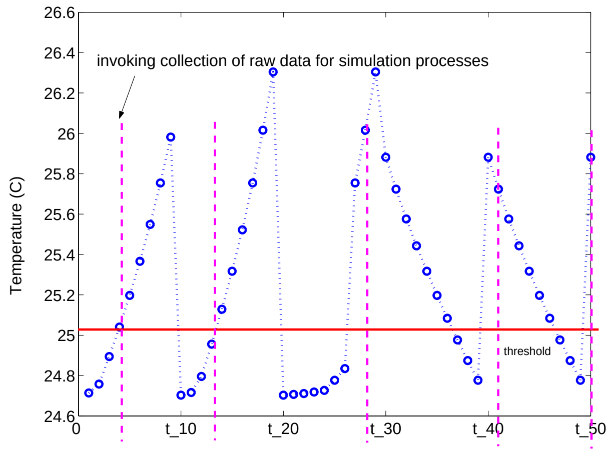


Figure 7.6: Maximal temperatures at the data center over time with global centralized management.

In a typical scenario, temperature values are collected from the sensor network, and are integrated with simulation processes offline to predict temperature distribution and determine appropriate job allocation policies. The plot in Figure 7.6 illustrates the maximal temperatures in the data center over time after running such an global management process. The plot demonstrates that the process has to be repeated often, which is expensive, and as a result, this approach cannot respond quickly to changes.

On the other hand, in-network analysis can be used to trigger local adaptations, so that, for example, when temperature increases above a certain threshold, a corresponding job instance can be migrated. This can achieved by appropriately specifying the reaction field of the *GridMap* operators, and locally collecting and processing (temperature) measurements, evaluating reaction rules and performing the required runtime job reallocation/migration.

The plot in Figure 7.7 compares the effect (in terms of the temperature at individual nodes) of using global (centralized) management, in-network local management, no management. It can be seen from this plot that with no management, the temperature of node with id 59 exceeds the redline threshold (i.e., 25 ^{o}C in this example). Furthermore, the effect of using in-network adaptations and the resulting temperature distribution (shown using the dotted line with circles) is comparable to using global centralized management (shown using the dotted line with stars).

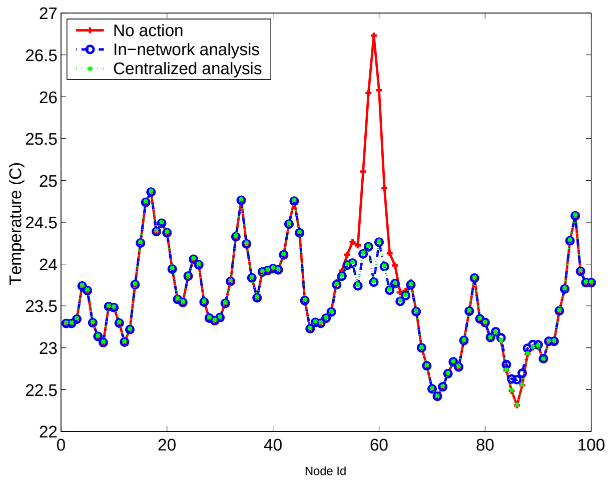


Figure 7.7: A comparison of the impact (in terms of node temperatures) of global centralized management, in-network management and no management.

Figure 7.8 plots the maximum temperatures in data center over a period of time with and without in-network adaptations. This plot demonstrates the effectiveness of using in-network management and its ability to response quickly to increases in the temperature. Figure 7.9 plots the communication overheads, in terms of number of messages, when using global centralized management versus in-network local management. The plot show that in-network management only uses 1/5th the number of messages (and consequently much less bandwidth).

The above results demonstrate the effectiveness of in-network local management, both in terms of reducing the costs of management and make it more responsive. These results validate our premise that it can be effectively used in conjunction with global management to achieve end-to-end autonomic management of instrumented data centers. We are currently continuing our experiments to further evaluate more realistic end-to-end scenarios.

7.5 Related Work

There exist three general ways of improving the energy consumptions and job management at the data center level. The first, which is the focus of this work, is to improve and optimize the cost from an end-to-end management point of view with the help of online data from the sensor network. The second is to improve and optimize the cooling costs, and especially the temperature distribution, during the operation of a data center. Finally, approaches for improving the

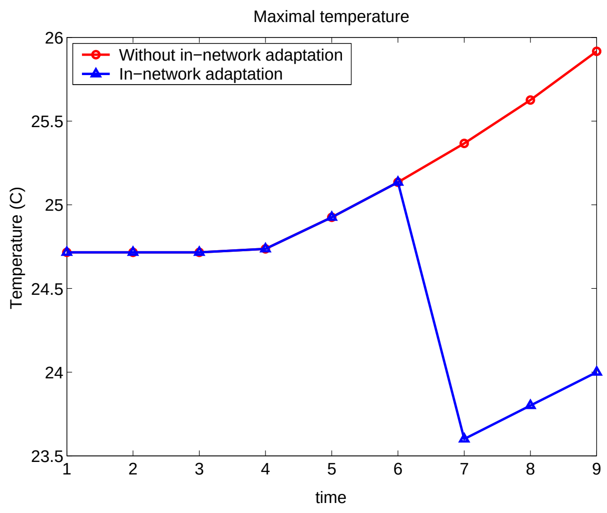


Figure 7.8: Maximum temperatures in the data center over time with and without in-network management and job migration.

power efficiency of application using various tradeoffs have been developed.

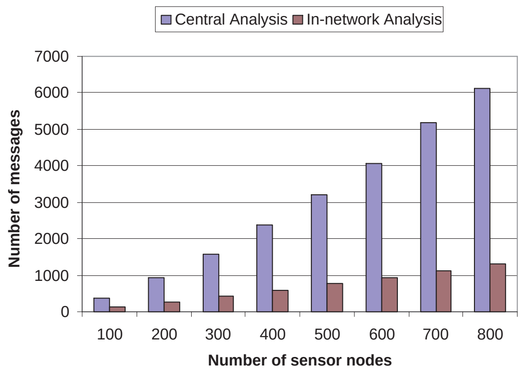


Figure 7.9: Communication overheads, in terms of number of messages, when using global centralized and in-network management.

Sensor networks for data center management. In [74], wireless sensor networks are deployed within a data center to collect data to understand heat distributions as a first step toward improving data center energy efficiency. Que [20] is a general rapid prototyping tool, and is developed to offer a scripting-based exploration environment and provide a simulation and emulation environment for a multi-platform tiered system. Our work is complementary to these approaches. After determining a centralized scheduling policy, we configure the sensor network with in-network policies as part of an end-to-end data center management system. Our goal to use local in-network management in conjunction with global management to reduce computation and communications overheads.

Making cooling more effective. Thermal-aware scheduling, proposed in [83], used simulation to conduct thermal evaluation. Machine learning methods are presented [84] to predict the effects of workload, cooling and airflow on the data center temperature distribution. In [111], a new policy called minimizing maximal inlet temperature is proposed to increase cooling efficiency for homogeneous datacenter management. However, an infrastructure for reacting to dynamically changing environments and managing emergencies also needs to model the thermal management policy itself at a fine grain, in terms of both temperature and performance, which is not considered in this work. This chapter used in-network policies to react to the dynamics in data center operation as part of a end-to-end autonomic management system.

Improving power efficiency using different tradeoffs. Several research efforts propose methods to jointly manage power and performance. Dynamic Voltage Scaling (DVS) [47] scales a processorś frequency, thereby reducing CPU performance and conserving system power. Another approach is to power-on only the minimal amount of server capacity as required for interactive applications [84]. A utility function is used in [60] to trade off response time with power saving for connection intensive web-based applications. C-Oracle [95] is a software prediction infrastructure that makes online predictions for a major class of data center-based systems, i.e., Internet services. Our work provides a two-level autonomic management system where at the higher level, global physical properties such as air flows and power consumption and utilization are used, and the at the lower level, a localized fast prediction model (e.g., with in-network policies) is used to adapt to changes when necessary. This differs from related efforts where global computations and predictions are required all the time.

7.6 Summary

This chapter proposed a two-level autonomic data center management system for instrumented data centers, described how it can be implemented using the *GridMap/iZone* programming system. Sensor networks monitor data center operating conditions such as temperature, humidity, and airflow, in real time, provide non-intrusive and fine-grained data collection, and enable online processing. The sensor data is integrated with computational processes and job schedulers to take phenomena such as heat distribution and air flows into consideration, and to optimize overall data center performance it terms of power consumption and job throughput. Experimental results show that the in-network analysis with the programming system achieve timely response and maintain near optimal management and control while reducing overheads.

Chapter 8

Summary, Conclusion and Future Work

8.1 Summary

The overall goal of this research is to develop sensor system middleware and programming support that will enable distributed networks of sensors to function, not only as passive measurement devices, but as intelligent data processing instruments, capable of data quality assurance, statistical synthesis and hypotheses testing as they stream data from the physical environment to the computational world. Further, application should be able to interact with the sensor system to control sensing and data processing behaviors.

The programming system enable sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms. The former supports complex querying of the sensor system, while the latter enables development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilations, both via semantically meaningful abstractions. Specifically for the latter, we explore the temporal and spatial correlation of sensor measurements in the targeted application domains to tradeoff between the complexity of coordination among sensor clusters and the savings that result from having fewer sensors for in-network processing, while maintaining an acceptable error threshold. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of constraint resources and satisfy data requirement in the presence of dynamics and uncertainty.

The research presented in this thesis is evaluated using two application scenarios: (1) the management and optimization of an instrumented oil field and (2) the management and optimization of an instrumented data center. In the first scenario, the programming abstractions and systems software solutions enable end-to-end management processes consisting of detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface. The overall goal is to ensure near optimal operation of the reservoir in terms of profitability, safety and/or environmental impact. In the second scenario, the autonomic instrumented data center management system addresses power consumption, heat generation and cooling requirements of the data center, which are critical concerns especially as the scales of these computing environments grow. Sensor networks monitor temperature, humidity, and airflow in real time, and provide non-intrusive and fine-grained data collection, and enable real-time processing. And these sensors are integrated with computational processes and job schedulers to take phenomenon, such as heat distribution and air flows into consideration, and to optimize data center performance in terms of energy consumption and throughput. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

In summary, this research investigated a programming system for sensor/actuate-driven applications by providing semantically meaningful abstractions and runtime mechanisms to seamlessly access and integrate a wide area sensor data into computational models and support scalable in-network data processing, such as aggregation, adaptive interpolation and assimilations.

8.2 Conclusion and Contributions

Technical advances are leading to a pervasive computational ecosystem that integrates computing infrastructures with embedded sensors and actuators, and giving rise to a new paradigm for monitoring, understanding, and managing natural and engineered systems – one that is information/data-driven. This research investigates a programming system to support the development of in-network data processing mechanisms, and enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems using semantically meaningful abstractions. This includes abstractions and runtime mechanisms for integrating sensor systems with computational models for scientific processes, as well as for in-network data processing such as aggregation, adaptive interpolation and assimilations.

This research provided a software infrastructure that enables experts to experiments with different aspects of end-to-end dynamic data-driven autonomic application and systems, including data acquisition, data assimilation and uncertainty management, data transport, and dynamic data injection. Specifically, the research is driven by the management and control of subsurface geosystems, such as the management and optimization of instrumented oil reservoirs. The programming abstractions and systems software solutions can enable the end-to-end management process for detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface, so as to ensure new optimal operation (in terms of profitability, safety or environmental impact). Another example is the autonomic instrumented data center management system for green computing. Sensor networks are used to monitor temperature, humidity, airflow, and energy-related data, computational processes and job schedulers in order to optimally manage energy and performance of data center. Such data center energy and performance management processes integrate real time monitoring data from sensor networks, computational processes of physics phenomenon to correlate and profile real-time input with end-user applications to optimize data center performance in terms of energy consumption and throughput. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

Key contributions of this research are summarized below.

The *GridMap/iZone* Programming System

The *GridMap/iZone* programming systems enable sensor-driven applications at two levels. First, it provides programming abstractions for integrating sensor systems with computational models for scientific processes (e.g. biophysical, geophysical processes) and with other application components in an end-to-end experiment. Second, it provides programming abstractions and system software support for developing in-network data processing mechanisms. The former supports complex querying of the sensor system, while the latter enables development of in-network data processing mechanisms such as aggregation, adaptive interpolation and assimilations, both via semantically meaningful abstractions. Specifically, the end-to-end abstractions provided by programming system is to enable scientific/engineering applications to discover, query, interact with, and control instrumented physical systems in a semantically meaningful way. For the latter, we explore the temporal and spatial correlation of sensor measurements in the targeted application domains to tradeoff between the complexity of coordination among sensor clusters. Experimental results show that the proposed in-network mechanisms can facilitate the efficient usage of resources and satisfy data requirement in the presence of dynamics and uncertainty.

A Decentralized Content-based Aggregation Service for Pervasive Grid Environments

Associative Rendezvous (AR) was as a paradigm for content-based decoupled interactions for pervasive grid applications. We also present Meteor, a content-based middleware infrastructure to support AR interactions. The aggregation service builds on the Meteor content-based middleware infrastructure and extends the Associative Rendezvous (AR) model for content-based information discovery and decoupled interactions. Specifically, it extends the AR abstractions to enable content-based aggregation queries to be flexibly specified using keywords, partial keywords and ranges. This also includes specification of recurrent and spatially constrained queries. Further, it builds on a self-organizing overlay network and the Squid content-based routing infrastructure to construct aggregation tries so that query propagation routes can be used for back-propagating and aggregating matching data elements. The deployment and experimental evaluation of the aggregation service are also presented. Evaluations include experiments using deployments on a LAN, the wireless ORBIT testbed [2] at Rutgers University, and the PlanetLab wide-area testbed [93], as well as simulations. Evaluation results demonstrate the scalability, effectiveness and performance of this decentralized aggregation service to support pervasive grid applications.

Enabling End-to-end Sensor-driven Scientific and Engineering Applications

The research presented in this thesis is evaluated using two application scenarios: (1) the management and optimization of an instrumented oil field and (2) the management and optimization of an instrumented data center. In the first scenario, the programming abstractions and systems software solutions enable end-to-end management processes consisting of detecting and tracking reservoir changes, assimilating and inverting data for determining reservoir properties, and providing feedback to enhance temporal and spatial resolutions and track other specific processes in the subsurface. The overall goal is to ensure near optimal operation of the reservoir in terms of profitability, safety and/or environmental impact. In the second scenario, the autonomic instrumented data center management system addresses power consumption, heat generation and cooling requirements of the data center, which are critical concerns especially as the scales of these computing environments grow. Sensor networks monitor temperature, humidity, and airflow in real time, and provide non-intrusive and fine-grained data collection, and enable real-time processing. And these sensors are integrated with computational processes and job schedulers to take phenomenon, such as heat distribution and air flows into consideration, and to optimize data center performance in terms of energy consumption and throughput. Experimental results show that the provided programming system reduces overheads while achieving near optimal and timely management and control in both application scenarios.

8.3 Future Work

The next decade of scientific discovery must be driven by advances in information technology. The sciences are fundamentally computational, which is also the essential challenges addressed by this research. The directions for future extensions of this research are envisioned as below.

Resource aware in-network computation in the presence of mobility. A range of in-network estimation mechanisms have been proposed with this research for aggregation, adaptive interpolation and assimilations in the presence of available resources and required quality. One extension is to develop resource aware mechanisms in the presence with mobility. For example, by using location aware infrastructure provided by the *GridMap/iZone*, in-network processing can make use of the changed location information and resource capability of mobile sensors around target area (e.g., due to the varied members and available resources of an *iZone*). Furthermore, the system should also address the challenge of the mobility, such as synchronization with in-network computation.

New application domains that use GridMap/iZone end-to-end programming system. The *GridMap/iZone* system can be used to develop, deploy, and experiment with sensor networks at scale in complex real-world outdoor urban environment. For example, for the city wide data-intensive, sensor-driven applications, an vehicle traffic and emergency surveillance system is useful for traffic control. The programming system for such applications should be location aware and mobility friendly to estimate traffic conditions from time to time. Our programming and runtime system can integrate the real-time monitoring system with advanced models and simulations to enable online data analysis and control with an end-to-end controller via feedback for vehicular traffic surveillance. Such controller can be implemented with by *GridMap/iZone* programming abstractions to enable the detection of natural or man-made events. Another example application domain is the aquatic observing systems, which can use *GridMap/iZone* programming for adaptive sampling based on application specific observations.

References

[1] Project JXTA. Internet: http://www.jxta.org.

[2] ORBIT testbed. Internet: http://www.orbit-lab.org/.

[3] Karl Aberer, Manfred Hauswirth, and Ali Saleh. Zero-programming sensor network deployment. *Next Generation Service Platforms for Future Mobile Systems (SPMS)*, January 2007.

[4] Ioannis Aekaterinidis and Peter Triantafillou. Internet scale string attribute publish/subscribe data networks. In *Proceedings of the ACM 14th Conference on Information and Knowledge Management (CIKM)*, Bremen, Germany, October 2005.

[5] United States Environmental Protection Agency. Report to congress on server and data center energy efficiency, 2007.

[6] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, and W. J. Kaiser. Wireless integrated network sensors: Low power systems on a chip. *Proceedings of the 24th European Solid State Circuits Conference*, 1999.

[7] Amol Bakshi, Viktor K. Prasanna, Jim Reich, and Daniel Larner. The abstract task graph: A methodology for architecture-independent programming of networked sensor systems. *Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services, EESR 05*, 2005.

[8] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS '05)*, Columbus, OH, June 2005.

[9] Nilanjan Banerjee, Jacob Sorber, Mark D. Corner, Sami Rollins, and Deepak Ganesan. Triage: Balancing energy and quality of service in a microserver. In *Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007)*, June 2007.

[10] Cullen Bash and George Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. *USENIX Annual Technical Conference 2007*, June 2007.

[11] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. *Technical Report, Computer Science Department, Stanford University*, 2003.

[12] Bryan Bayerdorffer. Distributed programming with associative broadcast. In *Proceedings of the 27th Annual Hawaii International Conference on System Sciences, Volume 2: Software Technology (HICSS94-2), Wailea, HW, USA*, pages 353–362, 1994.

[13] Ranjita Bhagwan, George Varghese, and Geoffrey M. Voelker. Cone: Augmenting dhts to support distributed resource discovery. *Technical Report, UCSD, CS2003-0755*, 2003.

[14] Matthew Brown, Seth Gilbert, Nancy Lynch, Calvin Newport, Tina Nolte, and Michael Spindel. The virtual node layer: A programming abstraction for wireless sensor networks. *ACM SIGBED Review*, 4(3):7–12, 2007.

[15] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. Virtual ring routing: network routing inspired by DHTs. *ACM SIGCOMM Computer Communication Review*, 36(4):351–362, September 2006.

[16] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, October 2001.

[17] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Fanklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. *In Proceedings of CIDR Conference*, 2003.

[18] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered DHT applications. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 97–108, 2005.

[19] Elaine Cheong, Judy Liebman, Jie Liu, , and Feng Zhao. TinyGALS: A programming model for event-driven embedded systems. *Proceedings of the 18th Annual ACM Symposium on Applied Computing (SAC'03)*, 2003.

[20] David Chu, Feng Zhao, Jie Liu, and Michel Goraczko. Que: A sensor netowrk rapid prototyping tool with application experiences from a data center deployment. *The 5th European Conference on Wireless Sensor Networks (EWSN 2008)*, 2008.

[21] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *International Workshop Design Issues in Anonymity and Unobservability*, 2001.

[22] Abhimanyu Das and David Kempe. Sensor selection for minimizing worst-case prediction error. *International Conference on Information Processing in Sensor Networks (IPSN'08)*, April 2008.

[23] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[24] Prabal Dutta, Mark Feldmeier, Joseph Paradiso, and David Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proceedings of Seventh International Conference on Information Processing in Sensor Networks (IPSN'08)*, April 2008.

[25] Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Cory Sharp, Jay Taneja, Gilman Tolle, Kamin Whitehouse, and David Culler. Trio: Enabling sustainable and scalable

outdoor wireless sensor network deployments. *The Fifth International Conference on Information Processing in Sensor Networks: Special Track on Sensor Platform, Tools, and Design Methods for Network Embedded Systems (IPSN/SPOTS '06)*, April 2006.

[26] Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, January 2002.

[27] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[28] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. *Proceddings of the 24th International Conference on Distributed Computing Systems (ICDCS'05)*, pages 653–662, 2005.

[29] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. A Scaleable Event Infrastructure for Peer to Peer Grids. In *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 66–75, Seattle, Washington, USA, 2002. ACM Press.

[30] Saurabh Ganeriwal, Chih-Chieh Han, and Mani Srivastava. Poster abstract: Spatial average of a continuous physical process in sensor networks. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[31] Deepak Ganesan, Deborah Estrin, and John Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *Proceedings of the ACM Workshop on Hot Topics in Networks*, pages 143–148, Princeton, NJ, USA, October 2002. ACM.

[32] Deepak Ganesan, Ben Greenstein, Deborah Estrin, John heidemann, and Ramesh Govindan. Multiresolution storage and search in sensor networks. *ACM Transactions on Storage (TOS)*, 1(3):277–315, August 2005.

[33] Deepak Ganesan, Sylvia Ratnasamy, Hanbiao Wang, and Deborah Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *2nd Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[34] Jun Gao and Peter Steenkiste. Rendezvous points-based scalable content discovery with load balancing. In *Proceedings of the Fourth International Workshop on Networked Group Communication (NGC'02), Boston, MA*, pages 71–78, October 2002.

[35] David Gay, Philip David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. *In Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation*, pages 1–11, 2003.

[36] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. IrisNet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, 2003.

[37] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. EmStar: a software environment for developing and deploying wireless sensor networks. *CENS Technical Report 34*, December 2003.

[38] Ben Greenstein, Eddie Kohler, , and Deborah Estrin. A sensor network application construction kit (SNACK). *Proceedings of 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*, November 2004.

[39] Gryphon: publish/subscribe over public networks. http://www.research.ibm.com/gryphon/papers/Gryphon-Overview.pdf.

[40] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programmingwireless sensor networks using *Kairos*. *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 126–140, June 2005.

[41] Abhishel Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. *Lecture Notes in Computer Science*, 3231:254–273, 2004.

[42] Gregory Hackmann, Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Agimone: Middleware support for seamless integration of sensor and ip networks. *Proceedings of 2006 International Conference on Distributed Computing in Sensor Systems (DCOSS '06)*, 2006.

[43] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys '05)*, pages 163–176, June 2005.

[44] Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. *Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03)*, 2003.

[45] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. *Proceedings of the Hawaii International Conference on System Sciences*, January 2000.

[46] Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalhos, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network Magazine Special Issue*, January 2004.

[47] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 38–48, 2003.

[48] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM Press, 2004.

[49] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, August 2000.

[50] JavaSpaces. http://www.javaspaces.homestead.com/.

[51] Nanyan Jiang and Manish Parashar. In-network data estimation mechanisms for sensor-driven scientific applications. *Proceedings of the 15th International Conference on High Performance Computing (HiPC)*, December 2008.

[52] Nanyan Jiang and Manish Parashar. Programming support for sensor-based scientific applications. *Proceedings of the Next Generation Software (NGS) Workshop in conjunction with the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2008.

[53] Nanyan Jiang and Manish Parashar. Enabling autonomic power-aware management of instrumented data centers. In *Proceedings of the Fifth Workshop on High-Performance, Power-Aware Computing in conjunction with the 23nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2009.

[54] Nanyan Jiang and Manish Parashar. Enabling end-to-end data-driven sensor-based scientific and engineering applications. In *Proceedings of the Dynamic Data Driven Application Systems (DDDAS 2009), in conjunction with the International Conference on Computational Science (ICCS 2009)*, May 2009.

[55] Nanyan Jiang, Cristina Schmidt, Vincent Matossian, and Manish Parashar. Content-based middleware for decoupled interactions in pervasive environments. Technical Report Number 252, Wireless Information Network Laboratory (WINLAB), Rutgers University, April 2004.

[56] Nanyan Jiang, Cristina Schmidt, Vincent Matossian, and Manish Parashar. Enabling applications in sensor-based pervasive environments. In *Proceedings of the 1st Workshop on Broadband Advanced Sensor Networks (BaseNets), San Jose, CA, USA*, October 2004.

[57] Nanyan Jiang, Cristina Schmidt, and Manish Parashar. A decentralized content-based aggregation service for pervasive environments. *2006 ACS/IEEE International Conference of Pervasive Services (ICPS)*, pages 203–212, June 2006.

[58] Sanem Kabadayi, Adam Pridgen, and Christine Julien. Virtual sensors: abstracting data from physical sensors. *Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 587 – 592, 2006.

[59] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan. Harvesting aware power management for sensor networks. In *Proceedings of 43rd Design Automation Conference (DAC)*, July 2006.

[60] Jeffrey O. Kephart, Hoi Chan, Rajarshi Das, David W. Levine, Gerald Tesauro, Freeman Rawson, and Charles Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. *The 4th IEEE Conference on Autonomic Computing*, June 2007.

[61] Hector Klie, W. Bangerth, X. Gai, Mary F. Wheeler, P.L. Stoffa, Mrinal Sen, Manish Parashar, U. Catalyurek, J. Saltz, and T. Kurc. Models, methods and middleware for grid-enable multiphysics oil reservoir management. *Engineering with Computers, Springer-Verlag*, 22:349–370, 2006.

[62] Kevin Klues, Vlado Handziski, Chenyang Lu, Adam Wolisz, David Culler, David Gay, and Philip Levis. Integrating concurrency control and energy management in device drivers. In *Proceedings of 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, October 2007.

[63] Venkata A. Kottapalli, Anne S. Kiremidjiana, Jerome P. Lyncha, Ed Carryerb, Thomas W. Kennyb, Kincho H. Lawa, and Ying Lei. Two-tiered wireless sensor network architecture for structural health monitoring. *SPIEs 10th Annual International Symposium on Smart Structures and Materials*, 2003.

[64] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: A framework for distributed data fusion. *Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys'03)*, pages 114–125, November 2003.

[65] Andreas Lachenmann, Pedro Jose Marron, Daniel Minder, and Kurt Rothermel. Meeting lifetime goals with energy levels. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[66] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring bandwidth between planetlab nodes. In *Proceedings of Passive and Active Measurement Workshop (PAM 2005)*, pages 292–305, Boston, MA, USA, March 2005.

[67] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, 2002.

[68] Philip Levis, David Gay, and David Culler. Active sensor networks. In *Proceedings of the Second USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*, 2005.

[69] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[70] Philip Levis, Sam Madden, David Gay, Joe Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in TinyOS. *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[71] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[72] Periklis Liaskovits1 and Curt Schurgers. Leveraging redundancy in sampling-interpolation applications for sensor networks. *Proceedings of the third International Conference on Distributed Computing on Sensor Systems (DCOSS)*, pages 324–337, 2007.

[73] Jie Liu, Maurice Chu, Juan Liu, James Reich, and Feng Zhao. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing*, 2(4):50–62, 2003.

[74] Jie Liu, Bodhi Priyantha, Feng Zhao, Chieh-Jan Mike Liang, Qiang Wang, and Sean James. Towards discovering data center genome using sensor nets. In *Proceedings of The Fifth Workshop on Embedded Networked Sensors (HotEmNets'08)*, March 2008.

[75] Jie Liu and Feng Zhao. Towards semantic services for sensor-rich information systems. *Second IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (Basenets 2005)*, 2005.

[76] Ting Liu and Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, June 2003.

[77] Konrad Lorincz, Bor rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. Resource aware programming in the pixie os. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[78] Liqian Luo, Tarek F. Abdelzaher, Tian He, and John A. Stankovic. EnviroSuite: An environmentally immersive programming framework for sensor networks. *ACM Transactions on Computational Logic*, 5(3):543 – 576, 2005.

[79] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation service for Ad-Hoc sensor networks. *in Proceedins of the USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[80] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database System*, 30(1):122–173, 2005.

[81] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[82] Vincent Matossian, Viraj Bhat, Manish Parashar, Malgorzata Peszynska, Mrinal Sen, Paul Stoffa, and Mary F. Wheeler. Autonomic oil reservoir optimization on the grid. *Concurrency and Computation: Practice and Experience, John Wiley and Sons*, 17(1):1–26, 2005.

[83] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. *In the 2005 Usenix Annual Technical Conference*, April 2005.

[84] Justin Moore, Jeff Chase, and Parthsarathy Ranganathan. Weatherman: Automated, online, and predictive thermal mapping and management for data centers. *Third IEEE International Conference on Autonomic Computing*, June 2006.

[85] Luc Moreau and Christian Queinnec. Resource aware programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(3):441–476, May 2005.

[86] Luca Mottola and Gian Pietro Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. *Proceedings of the second International Conference on Distributed Computing on Sensor Systems (DCOSS)*, pages 150–168, 2006.

[87] Ryan Newton, Arvind, and Matt Welsh. Building up to macroprogramming: An intermediate language for sensor networks. *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)*, April 2005.

[88] Ryan Newton and Matt Welsh. Region streams: Functional macroprogramming for sensor networks. *Proceedings of the First International Workshop on Data Management for Sensor Networks (DMSN)*, August 2004.

[89] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centers. *Science of Computer Programming*, 41:277–294, 2001.

[90] Santashil PalChaudhuri, Rajnish Kumar, Richard G. Baraniuk, and David B. Johnson. Design of adaptive overlays for multi-scale communication in sensor networks. *IEEE Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2006.

[91] Paolo Masci Paolo Corsini and Alessio Vecchio. Virtus: a configurable layer for post-deployment adaptation of sensor networks. *International Conference on Wireless and Mobile Communications, ICWMC '06*, 2006.

[92] Manish Parashar, Vincent Matossian, Hector Klie, Sunil G. Thomas, Mary F. Wheeler, Tahsin Kurc, Joel Saltz, and Roelof Versteeg. Towards dynamic data-driven management of the ruby gulch waste repository. *Proceedings of the Workshop on Dynamic Data Driven Applications and Systems, International Conference on Computational Science (ICCS)*, 2006.

[93] PlanetLab. Internet: http://www.planet-lab.org/.

[94] Andres Quiroz and Manish Parashar. Design and implementation of a distributed content-based notification broker for ws-notification. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 207–214, Barcelona, Spain, September 2006.

[95] Luiz Ramos and Ricardo Bianchini. C-Oracle: Predictive thermal management for data centers. *Proceedings of the Fourteenth International Symposium on High-Performance Computer Architecture (HPCA'08)*, October 2008.

[96] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, San Diego, California, United States, 2001. ACM Press.

[97] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, volume 3, pages 1664– 1669, 2005.

[98] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[99] Hans Sagan. *Space-Filling Curve*. Springer Verlag, May 1995.

[100] Cristina Schmidt and Manish Parashar. Flexible information discovery in decentralized distributed systems. In *Proceedings of the 12th High Performance Distributed Computing (HPDC)*, pages 226–235. IEEE Press, June 2003.

[101] Cristina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in p2p systems. *Internet Computing Journal*, 8(3):19–26, 2004.

[102] Cristina S. Schmidt. *Flexible Information Discovery with Guarantees in Decentralized Distributed Systems*. PhD thesis, Rutgers, the State University of New Jersey, October 2005.

[103] Mehdi Sharifzadeh and Cyrus Shahabi. Utilizing voronoi cells of location data streams for accurate computation of aggregate functions in sensor networks. *GeoInformatica*, 10(1):9–36, March 2006.

[104] Mitali Singh and Viktor K. Prasanna. A hierarchical model for distributed collaborative computation in wireless sensor networks. *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, 2003.

[105] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: A language and runtime system for perpetual systems. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[106] Chavalit Srisathapornphat, Chaiporn Jaikaeo, and Chien-Chung Shen. Sensor information networking architecture and applications. *International Workshop on Pervasive Computing*, 2000.

[107] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM'02, Pittsburgh, PA*, pages 73–86, August 2002.

[108] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM'01 Conference*, pages 149–160, San Diego, California, August 2001.

[109] Katalin Szlavecz, Andreas Terzis, Razvan Musaloiu-E., Joshua Cogan, Sam Small, Stuart Ozer, Randal Burns, Jim Gray, and Alexander S. Szalay. Life under your feet: An end-to-end soil ecology sensor network, database, web server, and analysis service. *MSR-TR-2006-90*, 2006.

[110] David Tam, Reza Azimi, and Hans-Arno Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. *Lecture Notes in Computer Science*, 2944:138–152, 2004.

[111] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transctions On Parallel and Distributed Systems*, 19(11):1458–1472, November 2008.

[112] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[113] Sameer Tilak, Paul Hubbard, Matt Miller, and Tony Fountain. The ring buffer network bus (RBNB) dataturbine streaming data middleware for environmental observing systems. *The 3rd IEEE International Conference on e-Science*, 2007.

[114] Robbert van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[115] Robbert van Renesse and Adrian Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. *The third international workshop on Peer-to-Peer systems*, 2004.

[116] Christopher M. Vigorito, Deepak Ganesan, and Andrew G. Barto. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *Proceedings of IEEE Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2007)*, pages 21–30, June 2007.

[117] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. *in Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, 2004.

[118] Geoff Werner-Allen, Stephen Dawson-Haggerty, and Matt Welsh. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[119] Geoff Werner-Allen, Konrad Lorincz, Mario Ruiz, Omar Marcillo, Jeff Johnson, Jonathan Lees, and Matt Welsh. Monitoring volcanic eruptions with a wireless sensor network. *Second European Workshop on Wireless Sensor Networks*, 2005.

[120] Kamin Whitehouse, Jie Liu, and Feng Zhao. Semantic streams: a framework for composable inference over sensor data. *The Third European Workshop on Wireless Sensor Networks (EWSN)*, February 2006.

[121] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: A neighborhood abstraction for sensor networks. *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MOBISYS'04)*, pages 99–110, June 2004.

[122] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, U.S.A., 1999.

[123] Peter Wyckoff. T Spaces. *IBM Systems Journal*, 37(3):454–478, 2001.

[124] Yong Yao and Johannes E. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, September 2002.

[125] Mohamed Younis, Moustafa Youssef, and Khaled Arisha. Energy-aware routing in clusterbased sensor networks. *International Symposium on Modeling, Anaysis and Simulationof Computer and Telecommunication Systems*, 2002.

[126] Feng Zhao, Jaewon Shin, and James Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.

Vita

Nanyan Jiang

2009 Ph.D. in Electrical and Computer Engineering; Rutgers University, NJ, USA.

2002 M.S. in Electrical and Computer Engineering; Rutgers University, NJ, USA.

1997 B.S. Beijing University of Posts & Telecommunications, Beijing, China.

2003-2008 Graduate Assistant, Center of Autonomic Computing & The Applied Software System Laboratory, Rutgers University, NJ, USA.

2006-2006 Summer Researcher, NEC Laboratories America, Princenton, NJ, USA.

1999-2002 Graduate Assistant, Winlab, Rutgers University, NJ, USA.

Selected Publications

Nanyan Jiang and Manish Parashar. Enabling end-to-end data-driven sensor-based scientific and engineering applications. *Proceedings of the Dynamic Data Driven Application Systems (DDDAS 2009), in conjunction with the International Conference on Computational Science (ICCS 2009)*, Baton Rouge, Louisiana, May 2009.

Nanyan Jiang and Manish Parashar. Enabling autonomic power-aware management of instrumented data centers. *Proceedings of the 5th Workshop on High-Performance, Power-Aware Computing (HPPAC), in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)*, Rome, Italy, IEEE Computer Society Press, May 2009.

Nanyan Jiang and Manish Parashar. In-network data estimation for sensor-driven scientific applications. *Proceedings of the 15th International Conference on High Performance Computing (HiPC)*, 2008.

Nanyan Jiang and Manish Parashar. Programming support for sensor-based scientific applications. *Proceedings of the Next Generation Software (NGS) Workshop in conjunction with the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Miami, Fl, IEEE Computer Society Press, April 2008.

Nanyan Jiang, Andres Quiroz, Cristina Schmidt and Manish Parashar. Meteor: A Middleware Infrastructure for Content-based Decoupled Interactions in Pervasive Grid Environments. *Concurrency and Computation: Practice and Experience*, John Wiley and Sons, (Online: DOI: 10.1002/cpe.1278), November 2007.

Nanyan Jiang, Cristina Schmidt, and Manish Parashar. A decentralized content-based aggregation service for pervasive environments. In *International Conference of Pervasive Services (ICPS)*, pages 203 - 212, 2006.

Nanyan Jiang, Cristina Schmidt, Vincent Matossian, and Manish Parashar. Enabling applications in sensor-based pervasive environments. In *Proceedings of the 1st Workshop on Broadband Advanced Sensor Networks (BaseNets)*, San Jose, CA, USA, October 2004.

Nanyan Jiang, Cristina Schmidt, Vincent Matossian, and Manish Parashar. Content-based middleware for decoupled interactions in pervasive environments. Technical Report Number 252, Wireless Information Network Laboratory (WINLAB), Rutgers University, April 2004.