**The Pennsylvania State University**
**The Graduate School**

# RESOURCE ALLOCATION IN INFORMATION-CENTRIC NETWORKS

A Dissertation in
Computer Science and Engineering
by
Fangfei Chen

© 2012 Fangfei Chen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2012

The dissertation of Fangfei Chen was reviewed and approved* by the following:

Thomas La Porta
William E. Leonhard Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Guohong Cao
Professor of Computer Science and Engineering

Piotr Berman
Associate Professor of Computer Science and Engineering

Anna Squicciarini
Assistant Professor of Information Sciences and Technology

Raj Acharya
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School.

# Abstract

In this proposed dissertation, we will study several resource allocation problems in information-centric networks (ICN). ICN is a novel networking architecture specifically designed for scalable and efficient distribution of information and content. This is different from the traditional Internet structure in that ICN focuses on delivery of information objects to users rather than end-to-end delivery of packets between hosts. Information flow in ICN starts from data collection and gathering, via processing and aggregation, and completes with dissemination and distribution. At each of these steps, different network resources serve the flows and provide a certain performance guarantee. However, resources are not infinitely available. Each type of resource is associated with a capacity and any allocation must not exceed this limitation.

This dissertation focuses on allocating network resources to meet a certain objective under complex and realistic constraints. Problems we consider include data dissemination in mobile and wireless mesh networks, replica placement in content distribution networks, convergecast in sensor networks and resource allocation with stochastic demands. For these problems, we seek efficient algorithms that both admit good performance guarantees and perform well in practice.

# Table of Contents

**List of Figures** viii

**List of Tables** x

**Acknowledgments** xi

**Chapter 1**
**Introduction** 1
1.1 Background . . . 1
1.2 Related Work . . . 2
1.3 Topics . . . 3
1.3.1 Proactive Data Dissemination to Mission Sites . . . 4
1.3.2 Time Slot Assignment to Mobile Users . . . 4
1.3.3 Replica Placement in Cloud-based CDNs . . . 4
1.3.4 Converge-cast and Aggregation in Sensor Networks . . . 5
1.3.5 Resource Allocation with Stochastic Demands . . . 5
1.4 Organization . . . 5

**Chapter 2**
**Proactive Data Dissemination to Mission Sites** 6
2.1 Introduction . . . 6
2.2 Related work . . . 8
2.3 Basic Problem formulation . . . 10
2.3.1 The Integer Programming (IP) formulation . . . 11
2.3.2 The cost function . . . 12
2.3.3 Approximate cost . . . 12
2.3.4 Hardness . . . 13

2.4 Heuristics for the basic problem . . . 14
2.4.1 Algorithm for centralized offline setting . . . 14
2.4.2 Algorithm for distributed online setting . . . 14
2.5 Performance evaluation . . . 15
2.5.1 Problem instance generation . . . 15
2.5.2 Results . . . 17
2.6 User-assisted dissemination . . . 19
2.7 Congestion . . . 22
2.7.1 Congestion models . . . 22
2.7.2 Virtual Occupation for the heuristics . . . 23
2.7.3 Evaluation . . . 25
2.8 Protocols . . . 28
2.8.1 Directed by the source . . . 29
2.8.2 Directed by the CN . . . 30
2.8.3 Overhead Simulation . . . 31
2.9 Conclusion . . . 31

**Chapter 3**
**Time Slot Assignment to Mobile Users** **33**
3.1 Introduction . . . 33
3.2 Related Work . . . 36
3.3 Problem Setting and System Architecture . . . 38
3.3.1 Problem Settings . . . 39
3.3.2 System Architectures . . . 40
3.4 Problem Model and Algorithms . . . 41
3.4.1 Problem Models . . . 41
3.4.2 IP Formulations . . . 42
3.4.3 Algorithms and Techniques for General Cases . . . 46
3.4.4 Optimal Algorithms for Special Cases . . . 50
3.5 Performance Evaluation . . . 52
3.5.1 Mobility Pattern and Data Generation . . . 52
3.5.2 Simulation with synthetic mobility patterns . . . 53
3.5.3 Simulation with Trace Files . . . 55
3.5.4 Varying Bandwidth . . . 56
3.6 Machine-dependent weights . . . 58
3.6.1 Degradation Functions . . . 58
3.6.2 Algorithm performance . . . 59
3.7 Conclusion . . . 63

**Chapter 4**
**Replica Placement Problem in Cloud-based CDNs** **64**
4.1 Introduction . . . . . . . . 64
4.2 Related Work . . . . . . . . 68
4.3 Problem Formulation . . . . . . . . 69
4.3.1 Problem settings . . . . . . . . 69
4.3.2 IP formulation . . . . . . . . 70
4.3.3 Basic operations . . . . . . . . 73
4.4 Offline Algorithms . . . . . . . . 74
4.4.1 Greedy Site (GS) . . . . . . . . 74
4.4.2 Greedy User (GU) . . . . . . . . 75
4.5 Online Algorithms . . . . . . . . 76
4.5.1 Greedy Request Only (GRO, dynamic) . . . . . . . . 77
4.5.2 Greedy Request with Preallocation (GRP, dynamic) . . . . . 77
4.5.3 Greedy Site Only (GSO, static) . . . . . . . . 77
4.6 Performance Evaluation . . . . . . . . 78
4.6.1 Potential replica sites, request pattern and cost . . . . . . . . 78
4.6.2 Performance of Offline Algorithms . . . . . . . . 81
4.6.3 Performance of Online Algorithms . . . . . . . . 84
4.7 Conclusion and Future Work . . . . . . . . 88

**Chapter 5**
**Convergecast and Aggregation in Sensor Networks** **89**
5.1 Introduction . . . . . . . . 89
5.2 Related Work . . . . . . . . 92
5.3 Network Model . . . . . . . . 93
5.4 One-shot Max-throughput . . . . . . . . 95
5.4.1 Problem Formulation . . . . . . . . 95
5.4.2 Hardness . . . . . . . . 96
5.4.3 Heuristics for the General Case . . . . . . . . 97
5.5 Min-length Periodic Scheduling . . . . . . . . 102
5.5.1 Problem Formulation . . . . . . . . 102
5.5.2 Hardness . . . . . . . . 103
5.5.3 Algorithms . . . . . . . . 104
5.6 Evaluation . . . . . . . . 109
5.6.1 One-shot Scheduling . . . . . . . . 109
5.6.2 Periodic Scheduling . . . . . . . . 111
5.7 Conclusion . . . . . . . . 112

**Chapter 6**
**Stochastic Resource Allocation** **114**
6.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114
6.2 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
6.3 Formulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
6.3.1 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . 118
6.3.2 A Simple Lower Bound . . . . . . . . . . . . . . . . . . . . . . . 119
6.3.3 Upper Bound . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
6.4 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
6.4.1 Effective Bandwidth Mapping Random Vectors to Vectors . 122
6.4.2 Effective Bandwidth Mapping Random Vectors to Scalars (**d-to-1**) . . . . . . . . . . . . . . . . . . . . . . . . . . 125
6.5 Numerical Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . 126
6.5.1 Experiment Setup . . . . . . . . . . . . . . . . . . . . . . . . . 126
6.5.2 Poisson Trials . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
6.5.3 Bernoulli Trials . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
6.5.4 Exponential Trials . . . . . . . . . . . . . . . . . . . . . . . . . 129
6.5.5 Relaxing the Constraints . . . . . . . . . . . . . . . . . . . . . . 129
6.5.6 Experiments with a Mix of Different Distributions . . . . . . 130
6.6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

**Chapter 7**
**Conclusions** **133**
7.1 General Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133
7.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134
7.3 Future Directions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136

**Bibliography** **137**

# List of Figures


2.1 Problem Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2 Six performance tests, varying the indicated parameters. . . . . . . 18
2.3 One-to-many sharing scheme. . . . . . . . . . . . . . . . . . . . . . 20
2.4 User-assistance tests, varying the indicated parameters. . . . . . . . 21
2.5 An example of how VO effects the placement of data items. . . . . . 25
2.6 Pull latency with and without VO. . . . . . . . . . . . . . . . . . . 26
2.7 Latency vs. $\alpha$ with and without VM. . . . . . . . . . . . . . . . . . 27
2.8 Messing-passage logic of two protocols. . . . . . . . . . . . . . . . . 29
2.9 Amount of overhead, as the indicated parameter varies. . . . . . . . 31

3.1 Scenario: clients encountering BSs. . . . . . . . . . . . . . . . . . . 34
3.2 Contact Time Windows . . . . . . . . . . . . . . . . . . . . . . . . 38
3.3 Two Formulations . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.4 Two cases of conflicting jobs. . . . . . . . . . . . . . . . . . . . . . 49
3.5 Special traffic patterns. . . . . . . . . . . . . . . . . . . . . . . . . 54
3.6 Performance comparison, varying maximum job size. . . . . . . . . . 55
3.7 Histogram of time window size . . . . . . . . . . . . . . . . . . . . . 56
3.8 Performance comparison, increasing the number of jobs. . . . . . . . 56
3.9 Performance comparison, increasing the estimated bandwidth. . . . . 57
3.10 Decreasing by *arccot* . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.11 Test on machine-dependent weights, varying number of jobs . . . . 60
3.12 Test on machine-dependent weights, varying the maximum job size 61

4.1 Problem Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.2 CDF of the Optimal Costs in All Tests . . . . . . . . . . . . . . . . 80
4.3 Performance Statistics for Offline Static Algorithms . . . . . . . . . 81
4.4 Relative Cost for Offline Algorithms . . . . . . . . . . . . . . . . . . 82
4.5 Performance Statistics for Online Algorithms . . . . . . . . . . . . . 85
4.6 QoS Violation Ratios Statistics for Online Algorithms . . . . . . . . 86
4.7 Normalized Relative Cost for Online Algorithms . . . . . . . . . . . 87

5.1 Problem relations and combinations. . . . . . . . . . . . . . . . . . . . 91
5.2 An example scenario. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
5.3 Hardness reduction for one-shot with secondary interference. . . . . . 97
5.4 Bad example for MIS-based algorithm with LST. . . . . . . . . . . . 99
5.5 Interference patterns. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
5.6 Bad example for STG. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
5.7 NP-hard proof for min-length periodic scheduling. . . . . . . . . . . . 104
5.8 Independent edges blockable by another edge. . . . . . . . . . . . . . 108
5.9 Evaluating one-shot convergecast algorithms. . . . . . . . . . . . . . . 110
5.10 Evaluating periodic convergecast algorithms. . . . . . . . . . . . . . . 111

6.1 Poisson, $p = 15\%$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
6.2 Poisson, $p = 10\%$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
6.3 Bernoulli, $p = 15\%$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
6.4 Exponential, $p = 15\%$ . . . . . . . . . . . . . . . . . . . . . . . . . . 129
6.5 Impact of $p$ on effective bandwidth $\beta_p(X)$ . . . . . . . . . . . . . . . 130
6.6 Bernoulli, $p = 15\%$, $\beta_p(b) = b + 0.5$ . . . . . . . . . . . . . . . . . . 131
6.7 A mix of Poisson, Bernoulli and Exponential distributions, $p = 15\%$ 131

# List of Tables

2.1 Summary of Notations . . . . . 13

3.1 Comparison of System Architectures . . . . . 41
3.2 Table of Notations . . . . . 42
3.3 Start-time Formulation . . . . . 44
3.4 Time-indexed Formulation . . . . . 44
3.5 Complexity of Two Formulations . . . . . 45
3.6 Performance Comparison . . . . . 63

4.1 Problem Settings . . . . . 70
4.2 Summary of Notations . . . . . 73
4.3 Typical Storage Cloud Prices in 2010 . . . . . 79
4.4 Simulation Parameter Summary . . . . . 80

5.1 Summary of algorithmic results. . . . . . 112

# Acknowledgments

First of all, I have many to thank Dr. Thomas La Porta. He taught me not only about research in computer science, but also being a responsible and generous person.

I thank the members of my dissertation committee for their guidance and support.

I thank all my coauthors, including Yosef Alayev, Amotz Bar-Noy, Katherine Guo, Matthew Johnson, Murali Kodialam, Diego Pizzocaro, Alun Preece and Yan Sun.

Finally, I wish to thank my family and many friends for moral support during my years of graduate study.

Chapter 1

# Introduction

## 1.1 Background

Information flow in networked systems starts from data collection and gathering, via processing and aggregation, and completes with dissemination and distribution. At each of these steps, different network *resources* serve the flows and provide a certain performance guarantee. Resources belong to a variety of devices, such as end devices (e.g., mobile handhelds), sensors, networks (e.g., bandwidth) and services (e.g., from the cloud).

We consider the information flow in a wide range of networks, such as wireless sensor networks, mobile networks, and data center networks. The problems we consider are related to information-centric networking [1] (ICN), which is a novel networking architecture specifically designed for scalable and efficient distribution of information and content. This is different from the traditional Internet structure in that ICN focuses on delivery of information objects to users rather than end-to-end delivery of packets between hosts. One-to-many communication is a norm in ICN, where information is produced once, cached or stored and then copied and sent to multiple recipients. Related work includes Content Distribution Networks [2], data dissemination [3], IP Multicast [4]. In addition, many-to-one communication is also an important paradigm, especially in the data gathering phase, or when information from multiple sources is disseminated to a single user.

Networks of such type often seek minimum delay and maximum throughput of information delivery. However, at different steps or in different scenarios, re-

quirements are often customized. For example, minimizing the time to deliver all data items or maximizing the number of data items delivered in time both improve the throughput, but lead to completely different abstract problems. Many such variations of a problem are interesting, important and widely applicable in both practice and theory.

Meanwhile, resources are not infinitely available. Each type of resource is associated with a capacity and any allocation must not exceed this limitation. To support networked applications, we need to satisfy their resource requirements in multiple dimensions within the capacity constraint of each dimension. Furthermore, inherent limitations of types of networks must be respected. For example, due to the nature of wireless communication, interference prevents multiple interfering devices from transmitting simultaneously, thus limiting the system performance.

This dissertation focuses on allocating network resources to meet a certain objective under complex and realistic constraints. In other words, we seek opportunities to optimize the utilization of network resources. To settle the competition for resources, many approaches have been proposed which generally fall into two categories: contention-based or scheduling/assignment. This dissertation emphasizes the second category which has the advantages of guaranteed performance (e.g. Quality of Service requirements) and low overhead.

## 1.2 Related Work

In this section, we discuss related work at a high level. Related work on each specific topic is covered in each chapter. Our work relates to a body of work on Content Distribution Networks (CDNs) and data dissemination.

A CDN consists of a group of servers that try to offload work from origin servers by delivering their content to other servers close to expected data request originators [2]. The goal of a CDN is to reduce the latency for a user requesting data. A well studied problem in CDNs is the Replica Placement problem, in which replicas of a Web site are efficiently placed in the network in order to minimize the storage and bandwidth costs [5, 6, 7, 8].

In the networking literature, three canonical data dissemination approaches

have appeared for sensor networks. External Storage stores all event data at an external storage point so queries for data need to go out of the network; Local Storage (e.g., Directed diffusion [3]) stores all event information locally so queries for data have to flood the network to find the data source; and Data-Centric Storage (e.g., Geographic Hash Table [9]) stores different types of data within the network at designated nodes; queries for data can then be sent directly to the node without flooding the network.

Theoretically, many underlying optimization problems relate to well studied problems, such as the Generalized Assignment Problem (GAP), Knapsack Problem, Bin Packing and Facility Location [10]. Considering a time factor, problems addressed in this dissertation are considered scheduling problems [11], especially job scheduling on parallel machines. Furthermore, this dissertation concerns some stochastic allocation problems such as the stochastic knapsack problem [12, 13].

In addition to the two broad areas of research, our work also relates to topics such as multicast. The benefit of using multicast rather than traditional multiple unicast links is that the source and many intermediate nodes only need to transmit the data once, which saves network resources. Application Layer Multicasting (ALM) [14] is closely related to our work, in the way that it constructs a multicast tree at the application layer without making assumptions about the underlying network structure.

## 1.3 Topics

This section introduces the problems studied in this dissertation and discusses their relations. The first part of the dissertation concentrates on data dissemination. We consider different problem settings and data delivery methods. We then consider data gathering, as information flows in the opposite direction. Finally, we extend our work on resource allocation to capture the case where resource requirements are stochastic.

### 1.3.1 Proactive Data Dissemination to Mission Sites

The dissertation starts with a data dissemination problem in wireless mesh networks, in which information is pushed to storage nodes near the expected destination of mobile users. Taking advantage of the prior knowledge of users' destinations, information is ready at the storage nodes before users arrive, thus reducing the retrieval latency. The goal is to minimize the latency-based costs with respect to the storage capacity of storage nodes. The abstract problem turns out to be a general assignment problem, thus we provide heuristics, which are then evaluated via simulation using real user mobility traces.

### 1.3.2 Time Slot Assignment to Mobile Users

In the second scenario, information is disseminated more proactively. Knowing the path and travel schedule of a mobile user, requested information is placed along the path and can be downloaded by users as they are moving. Because of the limited wireless transmission range between the user and the base station, information can only be downloaded within a time window. With many users competing for the down links of the base stations, the goal is to maximize the number of completed downloads in the system. We study this setting as a parallel-machine scheduling problem with the little-studied property that jobs may have *machine-dependent* time windows.

### 1.3.3 Replica Placement in Cloud-based CDNs

A replica site in CDNs is similar to the storage node in the first problem. However, unlike the first problem, the only item a replica site stores is the whole Web replica. Therefore, all users can share the same site. Rather than considering the storage capacity constraints, in this CDN problem, we enforce a Quality of Service distance between a user and the replica site where this user is served; a user should be close enough to a replica site to meet the QoS constraints. The challenge is how to pick and provision replica sites under a pay-as-you-go cost structure. We develop a suite of algorithms and evaluate them using the real Internet topology and web access traces.

### 1.3.4 Converge-cast and Aggregation in Sensor Networks

In this work, we move from a one-to-many problem setting to a many-to-one setting. Converge-cast is a many-to-one communication paradigm that concerns information flow in the opposite direction. Instead of disseminating information to multiple recipients, converge-cast collects data from multiple data sources, combines and aggregates data en-route when possible, then gathers the data at a base station. Nodes form a multi-hop wireless network, forward data and observe interference. In this scenario, we study two TDMA scheduling problems—maximizing throughput and minimizing schedule period, and provide guaranteed approximation algorithms. Specifically, we consider converge-cast in wireless sensor networks in this work.

### 1.3.5 Resource Allocation with Stochastic Demands

The last problem we consider is an extension to the previous works. In previous work, resource demands or information needs are known a priori or fixed once revealed. In this work, we study a general problem of allocating multiple resources among a group of users with stochastic demands. The goal is to admit as many users as possible to the system without violating the resource capacity more often than a predefined overflow probability. The problem is modeled as a stochastic multi-dimensional knapsack problem. We extend and apply the concept of effective bandwidth in order to solve this problem efficiently. Via numerical experiments, we show that our algorithms achieve near-optimal performance with specified overflow probability.

## 1.4 Organization

The rest of the dissertation is organized as follows: Chapter 2 and Chapter 3 present the work on data dissemination to the destination and en-route, respectively. Chapter 4 studies a similar problem but in the context of CDNs and the cloud. Chapter 5 concerns the opposite direction of information flows: converge-cast. Finally, Chapter 6 studies the stochastic resource allocation problem. We conclude in Chapter 7.

Chapter 2

# Proactive Data Dissemination to Mission Sites

This chapter studies the problem of assigning data to storage nodes near expected destinations of mobile users.

## 2.1 Introduction

Timely dissemination of information in wireless mesh networks required to execute missions is critical in many circumstances. In an example scenario, a search and rescue mission is commenced by personnel who make use of various kinds of information, e.g., maps, road conditions, medical records. Each class of personnel requires access to some (possibly overlapping) subset of the available information, which should be made available to them as quickly as possible upon their arrival at the mission site. For this purpose, data may be pushed to storage nodes at or near the mission site where it can be retrieved by personnel.

This setting differs from other content distribution environments in which data is pushed to storage nodes on a regular basis. In the applications considered here, the need for data at a specific location arises suddenly with the creation of a mission. Since we push the data only once the information requirements are clear, we can reduce latency costs and congestion by pushing only the data actually needed.

The goal of this chapter is to minimize the latency-based *cost* incurred by the

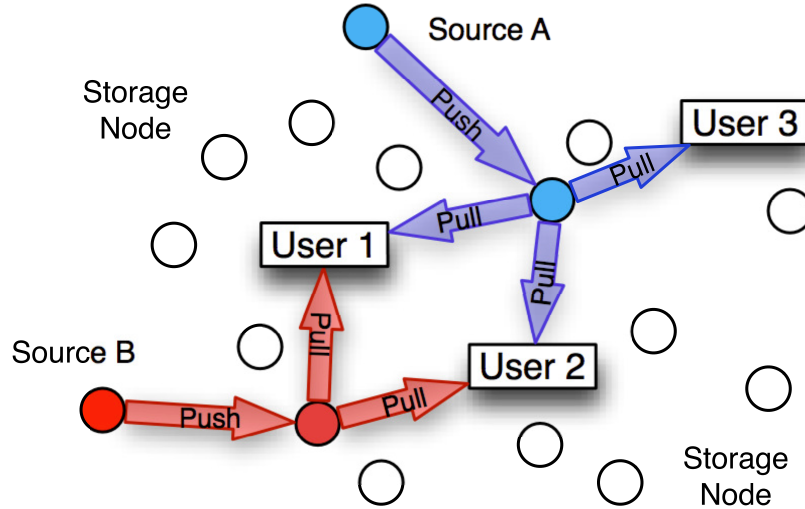



Figure 2.1: Problem Scenario

ultimate recipient when retrieving the data, while respecting the limitations on storage node capacities. Note that if the users are already at the mission site when data becomes ready, the latency experienced by them is the sum of the push and pull latency. If the users are far from the mission site when data becomes ready, the data may be pushed before the users arrive at the mission site. In this case, the latency to the users is only the pulling latency. We will show how our algorithms handle both these situations.

The network model (see Figure 2.1) is organized as follows: 1) a mission site lies within a wireless mesh network; 2) data sources push required data into storage nodes within the network; and 3) personnel (data recipients) traveling to the mission site pull the data from these storage nodes upon arrival at the mission site. Since the number and storage capacity of storage nodes are both limited, intelligent decisions must be made in allocating storage space to the various personnel types. Also, in the context of wireless mesh networks, bandwidth and energy are scarce resources, so we desire our algorithms to be scalable and energy-efficient, with low overhead.

We start with a simple setting in which the push/pull cost is defined as a function of data item size and the distances between the data's source, destination, and storage nodes. A related problem, called the Data Placement Problem (DPP),

is studied in [15]. The goal in one variant of that problem, which is NP-hard even to approximate, is minimizing the sum of installation (push) costs and access (pull) costs. Although our problem formulation differs in some respects, the hardness result carries over to it. Since no constant approximation algorithms are possible unless P=NP, our goal is to find efficient and distributed heuristics which perform well in general cases.

We define heuristics that allow a trade-off between push and pull costs and show that in the general case, where the number of data items is bounded by the number of nodes, we can achieve average costs within 15% of the optimum.

We then consider a more realistic setting in which both the location of the storage nodes and the congestion experienced in pushing and pulling the data contribute to the latency. We refine our heuristics by explicitly taking congestion into account and show that we can effectively trade-off pull and push costs, depending on the type of mission. Finally, putting the algorithms into the context of wireless mesh networks, we develop a fully distributed protocol and show that it is scalable with little overhead.

The rest of this chapter is organized as follows. Section 2.2 presents related work. Section 2.3 formally defines the basic problem and Section 2.4 presents the algorithms we use to solve the problem approximately. The algorithms are evaluated by simulations in Section 2.5. In Section 2.6, users' own storage capacity is used to assist with data dissemination. Section 2.7 extends the algorithms to account for congestion. Section 2.8 presents protocols to implement these algorithms. Section 2.9 concludes the chapter.

## 2.2 Related work

This work relates to a body of work on Content Distribution Networks (CDNs) and data dissemination. A CDN consists of a group of servers that try to offload work from origin servers by delivering their content to other servers close to expected data request originators [2]. The goal of a CDN is to reduce the latency for a user requesting data. Our model differs from a typical CDN in that the information needs in our system arise suddenly, and the location of the information requestors is only known once the data need arises.

Baev et al. [15] formulated and gave approximation algorithms for a related but simpler variant of the problem we consider. A similar problem in the area of Internet data requests (specifically, using the IP Anycast protocol), with minimizing time for "redirection" of a data item among the goals, was recently studied by Alzoubi et al. [16]. It also relates to delay-sensitive multicast routing [17, 18], where delay is considered as a constraint while the cost of a routing path is minimized.

In the networking literature, three canonical data dissemination approaches have appeared for sensor networks. External Storage (ES) stores all event data at an external storage point so queries for data need to go out of the network; Local Storage (LS) stores all event information locally so queries for data have to flood the network to find the data source; and Data-Centric Storage (DCS) stores different types of data within the network at designated nodes; queries for data can then be sent directly to the node without flooding the network.

Directed diffusion [3] is an example of LS, in which a source stores data locally and sends data only when a sink has sent a query for the data. One example of DCS is a Geographic Hash Table (GHT) [9], which hashes key values (such as data names) into geographic coordinates and stores the data at the sensor node geographically nearest to the hash value. The solution we present is most similar to DCS but differs in that we choose storage nodes based on a cost value. The dissemination is *proactive* in that we push data to these storage nodes ahead of time, so that the user can pull the data when needed at the mission site. Our goal is to minimize the total cost of pushing and pulling and to reduce latency.

At the core of our problem is the task of selecting the correct storage nodes. This requires assigning data items of various sizes to the best nodes. The core optimization problem here is the well studied Generalized Assignment Problem (GAP) [10], which lies in a family of related problems including Multiple Knapsack, Bin Packing and Facility Location.

A natural greedy strategy for the maximization variant of GAP [19, 20] is to take bins one at a time and solve each near-optimally. This provides a $2+\epsilon$-approximation when the single-bin problem is approximated within $1+\epsilon$. [21] gave LP-rounding and local-search algorithms for GAP and its generalization the Separable Assignment Problem (SAP), and was motivated by a Distributed Caching Problem (DCP). In that problem, there are data items, caches, and requests; caches

have storage capacities and bandwidth limits. Therefore there are two sets of decisions: the assignment of data items copies to each cache and the assignment of each request to some cache (containing data of that type). They choose a request assignment profit based on the associated assignment cost. [15, 22] (see also [23]) studied a minimization caching problem similar to both DCP and our problem in which caches have integral (degree) bounds on the number of assigned data items and clients have numeric demands for data items. The objective is to minimize storage and access costs. They obtain an LP-rounding-based 10-approximation obeying the hard capacity constraints, but with the restriction that data items are unit-size.

Shmoys and collaborators [24, 25] gave approximation algorithms for various special cases and relaxations of GAP, including a polynomial-time (though LP-rounding-based) algorithm that either finds an assignment of cost at most $C$ *while possibly violating capacity constraints by as much as the maximum data item size* or determines that no such solution exists. The work [16] cited above uses this algorithm and then shifts assignments from overloaded nodes as needed. Recently, [26] gave bicriteria algorithms for a setting allowing capacity constraints to be violated slightly, optimizing both total cost and the amount of capacity violation. Given a hard constraint on either goal, the other can be taken as the optimization goal. Given a cost bound, the problem of minimizing the violation of capacities (or in terms of machine scheduling, minimizing makespan) is known to be NP-hard to approximate better than 3/2 [25]; when capacities are strictly enforced, it is NP-hard to approximate the cost minimization problem to any constant factor, as in our stated formulation.

In this work, we enforce capacities strictly, and we require efficient, combinatorial algorithms that do not require LP-rounding. We therefore develop heuristic algorithms and evaluate them experimentally through simulation.

## 2.3 Basic Problem formulation

In this section we introduce the basic problem. Our goal is to minimize the latency-based cost of the data dissemination, where cost is defined as a function of the size of the data and distance it must travel. We extend the basic problem to incorporate

congestion in Section 2.7.

The number of hops taken by the wireless mesh networks to deliver the data is a natural metric for latency. In our specific mission scenario, because we want to route to a specific area (similar to GHT), we use a geographic routing protocol [27, 28], which makes greedy choices based on geographical information without needing to flood the network. An analytical approach was introduced in [29] to bound the number of hops for a given source-to-destination Euclidean distance, which allows us to use distance as a proxy for hops and avoid having to compute optimal hop counts.

### 2.3.1 The Integer Programming (IP) formulation

Let $D = \{d_i : i = 1 \dots m\}$ be a set of data items and $C = \{c_j : j = 1 \dots n\}$ be a set of storage nodes. Each data item $d_i$ has a size $p_i$ and belongs to one of the sources $S = \{s_\ell : \ell = 1 \dots p\}$. Each storage node has a storage capacity $b_j$. The network is designed to benefit a group of users $U = \{u_k : k = 1 \dots q\}$, each of which must retrieve some subset $D_k \subseteq D$ of the data items from mission sites. We assume that their locations are known when the mission starts. They can retrieve their data items either from the sources directly, or from the storage nodes. Each potential placement of a data item $d_i$ in a node $j$ (not necessarily a storage node) is associated with some cost $c_{ij}$, which initially is taken to be simply a property of the pair $(i, j)$, i.e. the sum of the cost of placing item $d_i$ in node $j$ and the total costs of accessing it from there, for all users who desire it.

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} & & (2.1) \\ \text{s.t.} \quad & \sum_j x_{ij} = 1 & & \forall i, \\ & \sum_i p_i x_{ij} \le b_j & & \forall j, \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

The objective is to minimize the total costs, consistent with the storage capacities. For each pair $(i, j)$, the decision variable $x_{ij}$ indicates whether this assignment

is chosen. The first set of constraints indicate that each data item must be placed once. The second set of constraints indicate that each node's storage capacity is respected. If data items can be divided fractionally, then this problem can be solved optimally by Linear Programming (LP). In this case, the relaxed decision variable $x_{ij}$ indicates the fraction of $d_i$ that is placed in node $j$.

There must be enough nodes to store all items, because in the worst case, each data source is capable of accommodating all its own data items; users could simply pull every data item from its source. To implement this, each data source node has large enough capacity to support its own data items in the formulation. However, to prevent sources from storing others' data, we set $c_{ij}$ to be prohibitively high for each source $j$ and data item $d_i$ not originating at $j$.

### 2.3.2 The cost function

In the general case, the cost $c_{ij}$ to store data item $i$ on storage node $j$ has two components, the push cost $s_{ij}$ and the pull cost $r_{ij}$ (in total, for all users requiring this item). We use parameter $\alpha$ and $1 - \alpha$ ($0 \leq \alpha \leq 1$) to indicate the relative priorities of these two costs:

$$c_{ij} = \alpha \cdot s_{ij} + (1 - \alpha) \cdot r_{ij} \qquad 0 \leq \alpha \leq 1 \tag{2.2}$$

In cases in which users are far from the mission site, and hence will not begin pulling data for a much longer time than the push time, the push cost may not be important. In this case the pull cost is prioritized by setting $\alpha$ close to 0. If the users are at or near the storage nodes when the mission is started, the push and pull costs are equally important and we want their sum to be minimized, then the push cost must be balanced with the pull cost by setting $\alpha$ to 0.5.

### 2.3.3 Approximate cost

As noted above, we approximate the number of hops by distance. Let $d(x, y)$ be the distance between *objects* $x$ and $y$ (where objects may be nodes, users, and/or data sources). With $U_i$ denoting the subset of users requesting data item $i$, the

| Term | Definition | Term | Definition |
|---|---|---|---|
| $d_i$ | Data item $i$ | $s_l$ | Data source $l$ |
| $c_j$ | Storage node $j$ | $u_k$ | User $k$ |
| $c_{ij}$ | Cost to store $d_i$ on $c_j$ | $b_j$ | Capacity of $c_j$ |
| $\alpha$ | Priority on push cost | $\rho$ | VO parameter |

Table 2.1: Summary of Notations

cost function can be written as:

$$c_{ij} = \alpha \cdot d(i,j)p_i + (1-\alpha) \cdot \sum_{k:u_k \in U_i} d(j,k)p_i \tag{2.3}$$

If $j$ is the source of $d_i$, then $d(i,j) = 0$. If $\alpha$ is a constant for each provider, then the cost depends on four factors: the source it belongs to, the users who request it, the storage node, and its size. The notations used in this chapter are summarized in Table 2.1.

### 2.3.4 Hardness

A special case of this problem with (metric) access costs only (i.e., $\alpha = 1$) and unit weights was shown to be NP-hard in [15]. That paper showed a non-unit weight version of their problem to be NP-hard to approximate, by proving it NP-hard to determine whether a feasible solution exists, given capacity constraints, by reduction from the Partition problem. That result does not *directly* apply here since, as just stated, we assume a feasible solution always exists. Still, their hardness argument can be adapted as follows, which indicates that no constant-factor approximation algorithm exists unless P=NP.

**Proposition 2.3.1.** *Our problem is NP-hard to approximate.*

*Proof.* Given is an instance of Partition, i.e., a set $A$ of $n$ numbers such that $\sum_{a_i \in A} = 2S$. We construct a problem with two users and two unit-capacity storage nodes, all four objects located in the same position $P$; $n$ data items, with each data item $i$ of size $a_i/S$, whose providers are located somewhere other than $P$; and $\alpha = 1$. Then determining whether there exists a zero-cost solution solves the underlying Partition problem. □

Rather than seeking an optimal solution, therefore, our interest in this chapter is efficient algorithms that perform well on realistic problem instances.

## 2.4 Heuristics for the basic problem

In this section, we motivate and present greedy heuristics in centralized offline and distributed online settings for the basic problem.

### 2.4.1 Algorithm for centralized offline setting

In the centralized offline setting, all information regarding the node assignments is known, and the assignment decision is made by a centralized mechanism. For example, a centralized server has complete knowledge of the missions, data items, storage nodes and users, and is capable of publicly announcing the assignments. In cases where such a server cannot be assumed, the centralized offline solution serves as a baseline in evaluating distributed online performance.

Assuming there are $m$ data items, $n$ storage nodes and $p$ data sources, a cost matrix $C$ of size $m \times (n+p)$ is maintained by the centralized server. Each element $c_{ij}$ of this matrix is the cost associated with placing data item $i$ in node $j$. The algorithm works greedily: it repeatedly places a data item with the max-minimum cost to its best-fit location. That is, first we sort items in each row of this matrix in order of increasing cost; then we compare the first item in each row, and sort these rows in order of decreasing first items. Finally, we traverse this matrix from left to right and top to bottom, assigning data items to nodes greedily with respect to the capacity constraints (see Algorithm 1).

### 2.4.2 Algorithm for distributed online setting

In the distributed online setting, each data source has knowledge of its own data items only. In Algorithm 2, each source works independently, attempting to push its data to the nodes in a greedy manner. Instead of having the entire matrix $C$, each source only has several rows of the matrix which are associated with its own data. The sources send out requests for their data to storage nodes, and because they act independently, conflicts may occur at storage nodes. To manage

**Algorithm 1** Centralized offline

```
1: generate a matrix C consisting of values c_ij, calculated according to Equation
   2.3
2: for i = 1 to m do
3:    sort c_i1, c_i2, ..., c_i(n+p) in increasing order
4: end for
5: sort rows in the order of decreasing first column values
6: for i = 1 to m do
7:    for j = 1 to n + p do
8:       if j is the source of d_i then
9:          d_i stays at its source
10:         break
11:      else if p_i ≤ b_j then
12:         place d_i at node j
13:         b_j ← b_j − p_i
14:         break
15:      end if
16:   end for
17: end for
```

these conflicts, storage nodes follow a "first come, first served" rule. If the storage node has enough space, it will accept the storage request; otherwise it rejects the request, and the source has to continue looking greedily for a storage node until it is satisfied.

## 2.5 Performance evaluation

Here we present the evaluation results for the heuristics for the basic problem.

### 2.5.1 Problem instance generation

For each set of parameters tested, we generate random mesh grids and present results averaged over 10 trials. In each trial, $n$ storage nodes are deployed in the grid, each with capacity $maxCapacity$. We generate $m$ data items whose sizes are selected uniformly at random from $[1, Datasize]$. Each of these data items is randomly assigned to one of $p$ data sources whose locations are also randomly selected. The mission site is located in the center of the field, with $q$ users uniformly

**Algorithm 2** Distributed Online

```
1: for each d_i on the same source do
2:   sort c_i1, c_i2, ..., c_i(n+p) in increasing order
3:   for j = 1 to n + p do
4:     if j is the source of d_i then
5:       d_i stays at its source
6:       break
7:     else if p_i ≤ b_j then
8:       send request to j
9:       if request accepted then
10:        place d_i at node j
11:        b_j ← b_j − p_i
12:        break
13:      end if
14:    end if
15:  end for
16: end for
```

distributed in the field. Each data item is requested by a given user with probability $reqProb$. By default, we set the priority parameter $\alpha$ to be 0.2.

To test the performance of the algorithms, we conduct six series of simulations, varying the following parameters:

1. the size of the problem instance (all $n$, $m$, $p$ and $q$)
2. the size of the problem instance with constant $n$
3. the number of data items $m$ only
4. $reqProb$
5. $maxDatasize$
6. the push priority $\alpha$

To measure performance, we compare the total cost obtained by our heuristics to a lower bound on the value of the problem's optimal cost. The lower bound used is the optimal solution value $OPT_{LP}$ to the LP relaxation of Problem 2.1, i.e., an optimal solution to the computationally easier problem in which data items may be stored fractionally in multiple storage nodes. Note that we do not allow such

fragmenting of data items for storage, so $OPT_{LP}$ can only be smaller than the true optimal value $OPT_{IP}$, meaning that the ratio computed is actually a conservative estimate of the approximation ratio achieved. The size of this ratio indicates the quality of the solution, with a ratio of 1 corresponding to optimality.

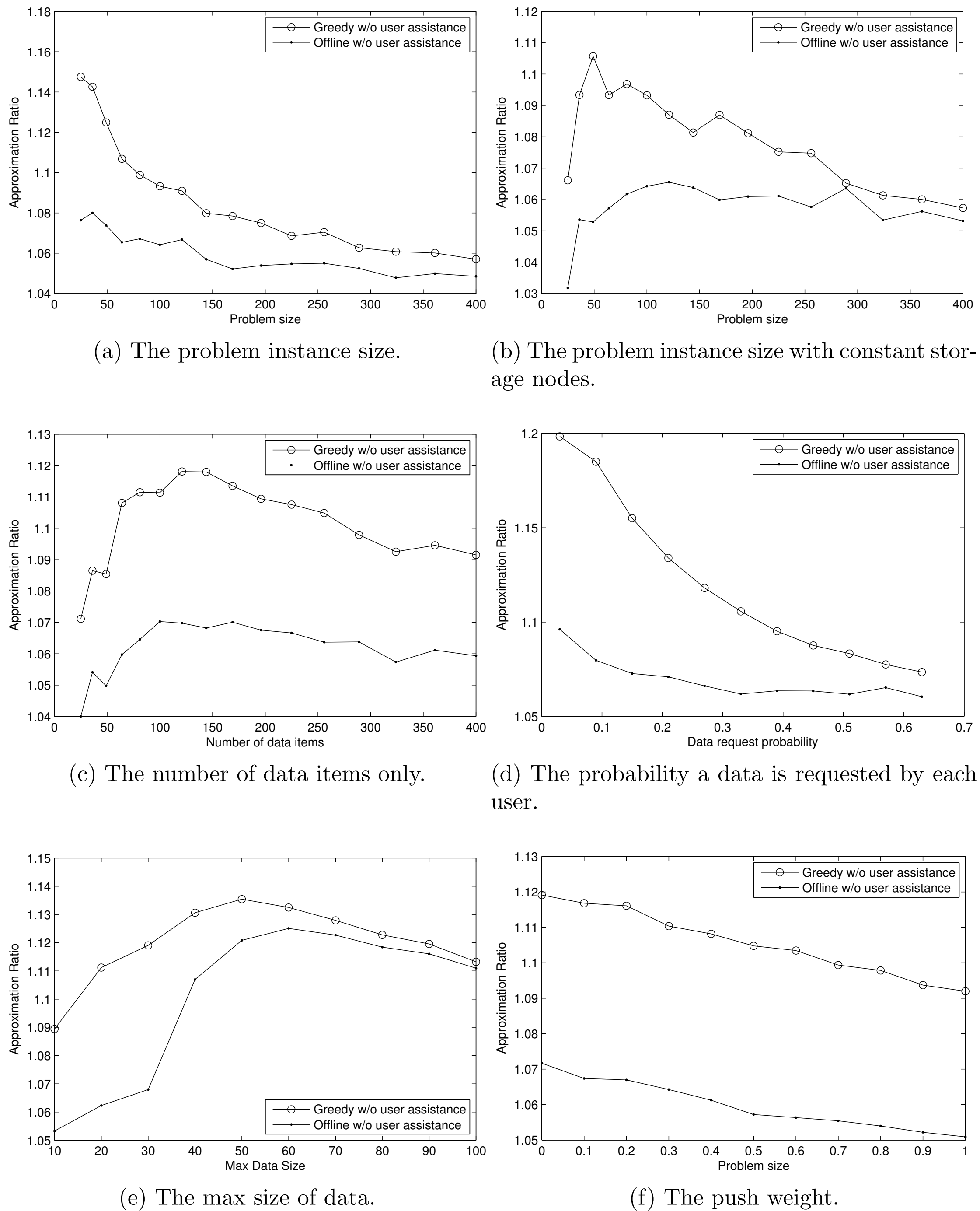
### 2.5.2 Results

The centralized offline algorithm consistently beats the distributed online algorithm. In spite of this, both algorithms achieve good approximation on average, in different situations.

In the first test, we increase the problem instance size, holding the ratios of data item, storage node, source and user counts fixed at $1 : 1 : \frac{1}{3} : \frac{1}{2}$. In Figure 2.2a, the offline approximation ratio is almost constant while the online ratio drops as the number of data items increases and approaches that of the offline. This is because as the number of data items increases, there are also more available storage nodes.

The second test is the same as the first, except that we hold the number of storage nodes fixed at 100. In Figure 2.2b, first we can see a fast increase of the approximation ratio when there is a small number of data items. As the number increases, the approximation ratio increases quickly before stabilizing. The ratio of the online algorithm drops slightly and gets very close to that of the offline algorithm. We interpret this behavior as follows: increasing the number of data items while fixing the number of storage nodes may lead to a higher approximation ratio; however, as more users request each data item (we did not fix the number of users in test 2), the (dominant) pull cost does not change much on different nodes. Also, as the number of data items increases, most data items have to stay at their sources, which does not increase the difficulty of the problem.

To corroborate the above explanation, we conducted tests 3 and 4. In test 3, we increase only the number of data items, holding other parameters constant. This removes the factor of increasing the number of users requesting the same data item. As shown in Figure 2.2c, the ratios of the two algorithms both increase and then stabilize. The ratio of the online algorithm never approaches that of the offline algorithm. In test 4, we increase the probability that a data item is requested by each user. This has the same effect as increasing the number of users requesting




(a) The problem instance size.

(b) The problem instance size with constant storage nodes.

(c) The number of data items only.

(d) The probability a data is requested by each user.

(e) The max size of data.

(f) The push weight.

Figure 2.2: Six performance tests, varying the indicated parameters.

each data item. As expected, in Figure 2.2d we can see the two approximation ratios both drop and converge in the end.

In the fifth test, we only increase the maximum size of data items. As we can see in Figure 2.2e, the two ratios first increase, then drop slightly, and finally converge. As the size of data items increase, the competition for storage nodes becomes severe. As the data sizes keep increasing, more data items will remain at their sources, so the competition becomes less intense.

In the sixth test, we increase the weight on the push cost. As we decrease the weight on pull, we put less focus on the mission site, so that the competition eases. As expected, in Figure 2.2f, both ratios decline.

To summarize, the following situations can degrade performance:

1. increasing the number of data items

2. increasing data sizes to a certain level

3. too few users requesting for each data item

4. too much weight on the pull cost

Both the distributed and centralized algorithms achieve good approximation in these different situations regardless of problem instance size.

## 2.6 User-assisted dissemination

In this section, we study how users' own storage capacity can be used to assist with data dissemination. In the scenario we consider, when we start to push data items some users may already be at the mission site. If these users are willing cooperate with other users by sharing data items they possess, then the presence of users with free storage space effectively increases the capacity of the network. The data items placed in a given user's storage may or may not be items that users requests. We assume this sharing is one-to-many. In Figure 2.3, for example, a source pushes its data item to one of the users who request it (User 0); other users then pull it from User 0.

The scheme fits in to our algorithms seamlessly. We simply extend the cost matrix with more columns (one for each user) and add more capacity constraints

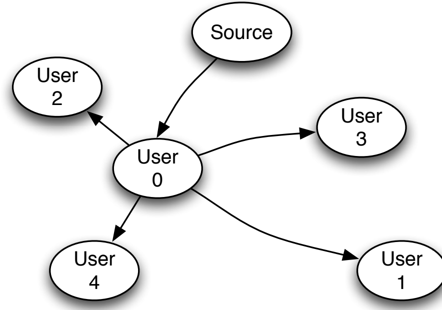



Figure 2.3: One-to-many sharing scheme.

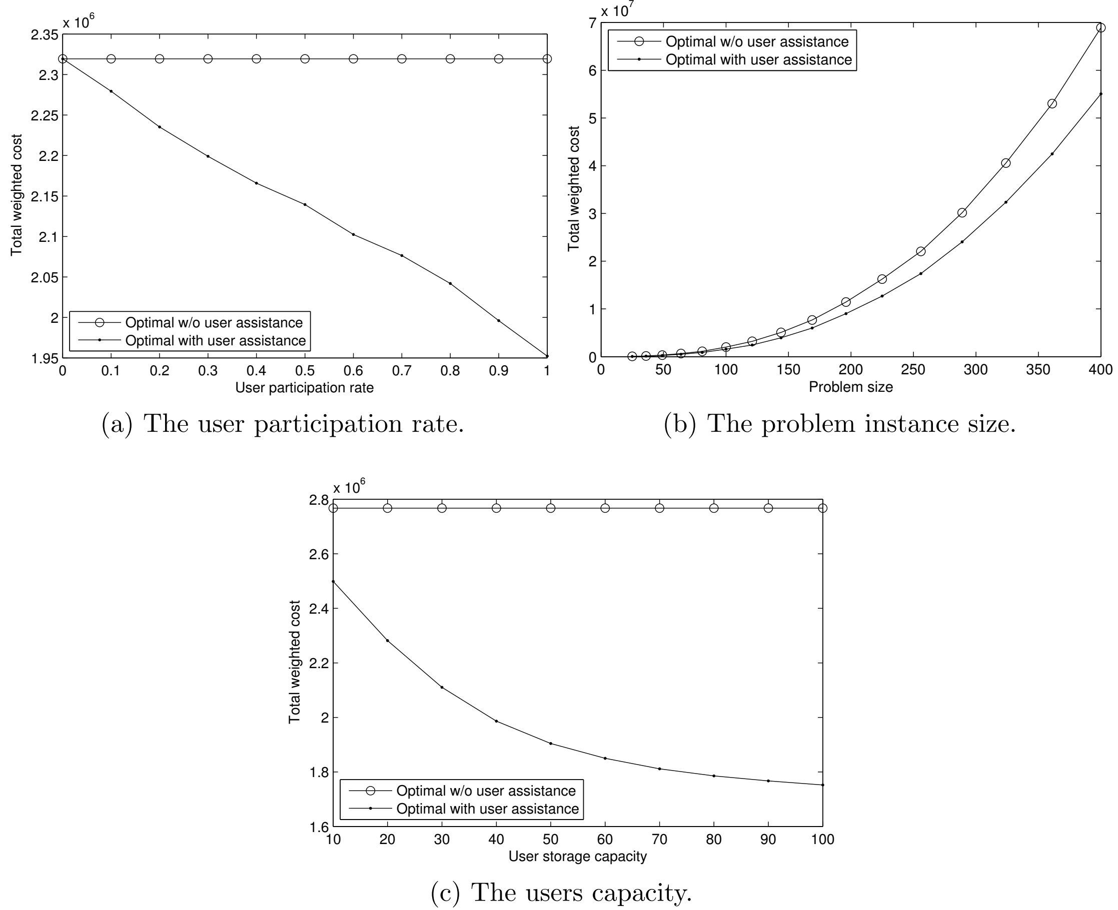
according to users' storage capacity. Equation 2.3 still applies to calculate the cost to store data on user nodes. Since cost depends on distance, the cost of a user pulling data from itself is 0.

This sharing will reduce the total cost, because the potential storage locations for data items are extended. Also, since users are likely to be physically close to each other, the extended storage space has more high value locations where the pull cost is low.

An important factor that affects the usefulness of this extension is the participation rate of users. Let the participation rate be defined as the proportion of users who are able and willing to share data items. In our one-to-many scheme, a user may be capable of sharing a data item if it is already at the mission site when the item is pushed.

To see how the participation rate affects the gains from such sharing, we conduct the following test: the scenario is generated similarly to those in Section 2.5 except that users have storage capacity of $userCapacity = maxDatasize$. The participation rate, $parRate$, is set to be the probability that a user participates in sharing data items. In Figure 2.4a, we plot the bound on the optimal cost values in problem instances with and without user assistance. As the participation rate increases, the cost value drops as expected.

In the second test, we increase the problem instance size and plot the bound on the optimal cost with and without user assistance. Note that in this test, the number of users increases as the number of data items increases, so that the user




(a) The user participation rate.

(b) The problem instance size.

(c) The users capacity.

Figure 2.4: User-assistance tests, varying the indicated parameters.

to data ratio stays constant. The result is shown in Figure 2.4b. The cost value with user assistance is always lower than the cost without, by about 25%.

Users generally have more storage capacity than the wireless nodes. To see how the effectiveness of the user assistance varies as the user capacity changes, in the third test we hold the size of the problem instance constant while increasing user capacities. In Figure 2.4c, we see that as users gain storage capacity, the cost decreases significantly and finally reaches a limit. There is a point beyond which the cost cannot be improved, however–the minimum sum of distances from all users to any node in the region.

In conclusion, either increasing the user participation rate or increasing the users' storage space for sharing will decrease the total cost. However, sufficiently

increasing users' storage capacity will eventually run up against a minimum barrier on the cost.

## 2.7 Congestion

In this section we extend heuristics to explicitly consider the challenge of congestion. This allows us to compare algorithms in terms of the latency experienced by users when retrieving the data.

### 2.7.1 Congestion models

Since latency is theoretically proportional to number of hops, minimizing the costs as discussed in Section 2.3 should tend in practice to minimize latency. Latency can also be increased, however, by congestion. For pushing data, the connection model is one-to-many: a source may push several data items to their storage nodes simultaneously. Congestion is likely to happen for three reasons: 1) the sources are positioned close to each other; 2) the storage nodes positioned are close to each other; or 3) the routes between the sources and storage nodes cross and interfere with one another. For pulling data, the model is many-to-one: several users may request data from the same node. In this case, congestion may occur for three similar reasons: 1) the users are close to each other; 2) the nodes from which data is being pulled are close together; or 3) the routes between the different storage and users pairs interfere with each other.

Congestion by cause 1 is inevitable because the sources are fixed in our problem. Cause 3 in both cases may be handled by specialized routing protocols which is beyond the scope of this chapter. We focus on addressing cause 2: how to pick storage nodes that are close to the ultimate pulling nodes considering the congestion that may be caused during the push and pull operations. Our basic solution is to discourage sources from picking storage nodes that are within close range of other heavily used nodes.

### 2.7.2 Virtual Occupation for the heuristics

To reduce this type of congestion, we want to limit the total size of data pushed to any given area. To do this, we extend our heuristics by introducing the concept of *Virtual Occupation* (VO). The goal of this extension is to disperse data around the field in order to avoid congestion, so that heavily used nodes do not lie too near to one another. When a storage node accepts data, we record that it occupies its own memory, as usual, but also "virtually" occupies some space on its neighbors. When neighboring nodes receive requests to store data, they treat the virtually occupied space as unavailable. This reduces the amount of data they can store, which naturally spreads data out, around the mission site. While this may increase the number of hops required for a user to pull data, it ideally will reduce congestion at the storage nodes and thus reduce overall latency. Note that this technique applies to both the centralized and distributed algorithms.

Algorithm 3, which is performed by each source, works as follows. When a node $j$ stores data item $d_i$ of size $p_i$, it reduces the available space in its one-hop neighbors by $\rho p_i$ ($\rho \in (0, 1]$) as seen by other data sources. Note that the available memory of the one-hop neighbors is not reduced with respect to the source of data item $d_i$ because a source will not interfere with itself, and although multiple users may request the same data item, e.g. $d_i$, this is less likely than different users requesting data items from different sources.

Similarly, for the centralized setting we give Algorithm 4, which is performed by the centralized server.

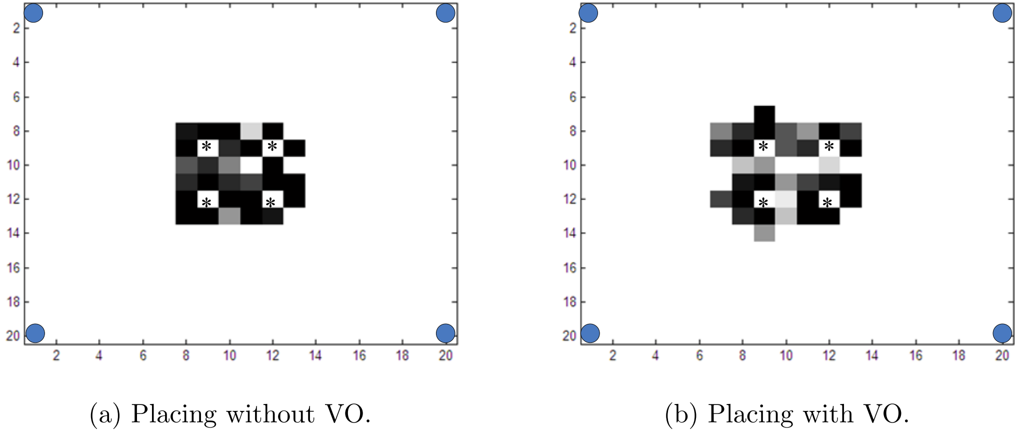
Figure 2.5 shows an illustrative example of the use of the virtual occupation algorithm. There are four source-user pairs. The sources are located at the four corners of the field, while the users are close together. The darker the shading of the squares in the figure, the larger the amount of data stored in the area. Without using the virtual occupation (Figure 2.5a), the area between the two users experiences heavy load (the darker squares). When using virtual occupation (Figure 2.5b), this area is less heavily used, which reduces the congestion among these source-user pairs.

**Algorithm 3** Distributed Online with VO

```
1: for each d_i on the same source do
2:    sort c_i1, c_i2, ..., c_i(n+p) in increasing order
3:    for j = 1 to n + p do
4:       if j is the source of d_i then
5:          d_i stays at its source
6:          break
7:       else if p_i ≤ b_j then
8:          send request to j
9:          if request accepted then
10:            place d_i at node j
11:            b_j ← b_j − p_i
12:            for each of k of j's neighbor do
13:               b_k ← b_k − ρp_i
14:            end for
15:            break
16:         end if
17:      end if
18:   end for
19: end for
```

**Algorithm 4** Centralized offline with VO

```
1: generate a matrix C, calculate c_ij according to Equation 2.3
2: for i = 1 to m do
3:    sort c_i1, c_i2, ..., c_i(n+p) in increasing order
4: end for
5: sort rows in the order of decreasing first column
6: for i = 1 to m do
7:    for j = 1 to n + p do
8:       if j is the source of d_i then
9:          d_i stays at its source
10:         break
11:      else if p_i ≤ b_j then
12:         place d_i at node j, b_j ← b_j − p_i
13:         for each of k of j's neighbor do
14:            b_k ← b_k − ρp_i
15:         end for
16:         break
17:      end if
18:   end for
19: end for
```




(a) Placing without VO.

(b) Placing with VO.

Figure 2.5: An example of how VO effects the placement of data items.

### 2.7.3 Evaluation

In this subsection we evaluate the virtual occupation algorithm and its effectiveness in mitigating congestion.

#### 2.7.3.1 Virtual Occupation

To see the effect of the virtual occupation clearly, we conducted the following tests in NS2: we fix the four source-user pairs as in Figure 2.5 but vary the number of requests from each user. The size of each data item is still uniformly distributed from 1 to 10. The latency results with and without VO are shown in Figure 2.6. The users request a varying number of data items as fast as possible; for example, a value of 5 on the X-axis in Figure 2.6 corresponds to each user requesting 5 data items. In this example we set $\alpha$ of Equation 2.3 to 0 to see the effects of the virtual occupation algorithm when the pull costs are emphasized. We tune the parameter $\rho$ (as in Algorithm 3) from 0 to 1.

We see that when the request load from each user is low, the virtual occupation technique does not improve performance because there is little congestion near the storage nodes. When the rate of requests increases, in most cases the virtual occupation technique reduces the pull latency. In some cases the reduction in latency is dramatic (40%). Even with high loads, however, in some cases (e.g., with 11 requests) the virtual occupation technique does not improve performance.

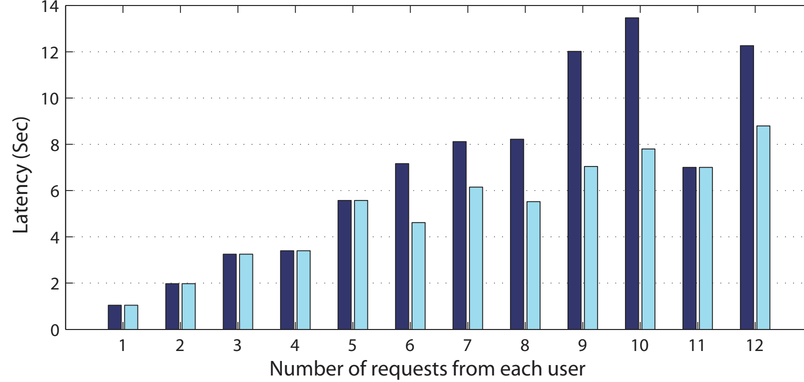



Figure 2.6: Pull latency with and without VO.

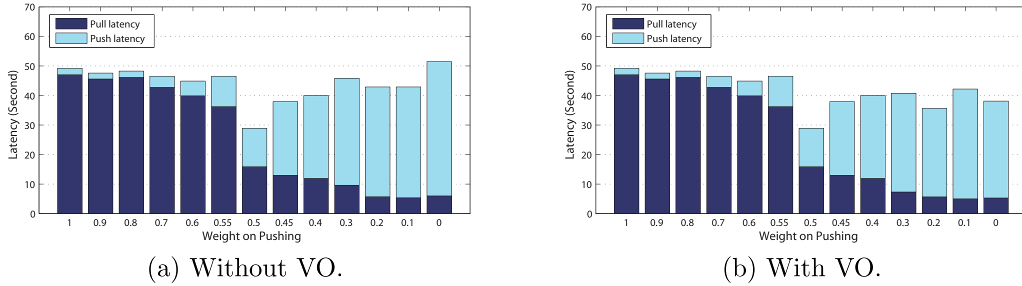
This is because the virtual occupation technique may increase the number of hops required to retrieve data to a point that the additional latency caused by the increased number of hops outweighs the benefits of reduced congestion.

#### 2.7.3.2 Trade-off between pushing and pulling

As seen in Equation 2.3, a trade-off may be made between the push and pull costs by adjusting the weight $\alpha$. Similarly, putting different weights on pushing and pulling will cause different latencies. This trade-off is important when considering different application scenarios. In general, the latency $T_r$ in retrieving data as seen by a user may be expressed as:

$$T_r = \begin{cases} T_{ps} - T_u + T_{pl} & \text{if } T_{ps} > T_u \\ T_{pl} & \text{otherwise} \end{cases} \tag{2.4}$$

where $T_{ps}$ is the time to push the data, $T_{pl}$ is the time to pull the data and $T_u$ is the movement time for the user to the mission site. Note that if the movement time of the user is longer than the push time of the data, it will not experience any push delay because it cannot retrieve the data until it arrives at the mission site. In this case the latency experienced by the user is just the pull latency $T_{pl}$. If the user is at the mission site when the mission arrives, $T_u$ is zero and the latency experienced by the user is a combination of the push time and pull time. If the user arrives during the time when the data is still being pushed, then the latency experienced by the user is a combination of a portion of the push cost and the full




(a) Without VO.

(b) With VO.

Figure 2.7: Latency vs. $\alpha$ with and without VM.

pull cost.

To evaluate the ability of our algorithms to balance this trade-off, we conducted a series of simulations in NS2 using an 802.11 network in which we use one fixed scenario as in Figure 2.5 while changing $\alpha$ from 1 to 0, i.e., we vary the cost smoothly from depending solely on push to depending solely on pull. The number of requests from each user is 12.

The non-VO latency results are shown in Figure 2.7a. As $\alpha$ decreases, the push latency increases and the pull latency decreases, as expected. We also notice that the the sum of the two latencies decreases as $\alpha$ decreases from 1 to 0.5, and then increases. The effect of congestion is also evident in this figure. Note that the pull latency rises slightly as $\alpha$ is decreased from 0.1 to 0; this is a result of increased congestion at the storage nodes when there is a high number of requests.

We also evaluated the impact of the $\alpha$ value with VO turned on (see Figure 2.7b). The parameter $\rho$ (as in Algorithm 3) is tuned from 1 to 0. In these simulations, sources are located far from one another while the users are relatively close to one another. Therefore the primary expected cause of congestion is closely spaced storage nodes, i.e., the second congestion type discussed in section 2.7.1. When $\alpha$ is large ($> 0.5$) and the data is not pushed all the way to the mission site, the chosen storage nodes are likely to lie near the sources and there will be little congestion of this type. When $\alpha$ is smaller, the storage nodes will tend to lie near to the mission sites and hence close to one another, in which case this type of congestion is likelier to appear.

In this case, the virtual occupancy algorithms work to reduce the congestion, and hence latency, for both push and pull. The reduction in push latency as $\alpha$

decreases in Figure 2.7b as compared to Figure 2.7a is clear, as is the reduction in total latency (e.g., when $\alpha = 0$, the total latency is reduced from over 50 seconds in Fig 2.7a to under 40 in Fig 2.7b). There is also reduction in the pull latency (as shown in bar 12 of Figure 2.6), but this is not as visible in Figs. 2.7b and 2.7a due to scaling. We observe that the pull latency does not reduce much when the virtual occupation algorithm is used as $\alpha$ decreases beyond 0.2. This is because although the reduction in $\alpha$ drives data closer to the mission site, the virtual occupation algorithm is preventing the data from completely reaching the mission site, thus maintaining the lowest pull latency, unlike the case in Figure 2.7a in which the pull latency increases slightly when $\alpha = 0$.

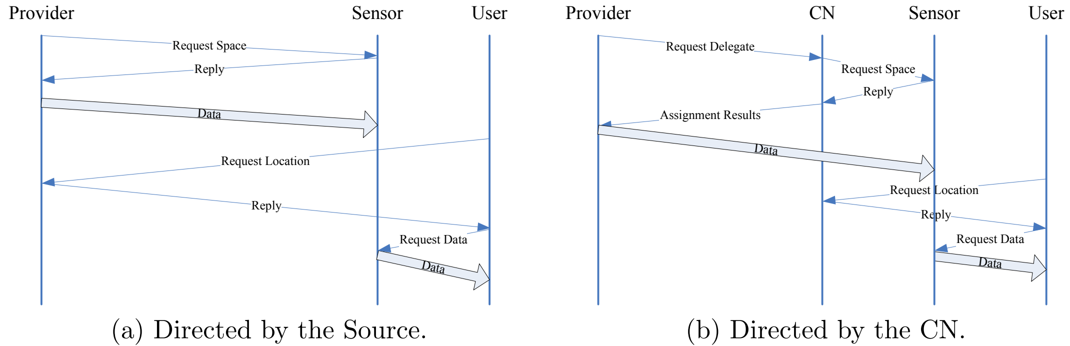
The the sum of push and pull latency is minimized when $\alpha = 0.5$. We draw the following conclusions. First, if a user has a longer time to travel to the mission site than the push time, i.e. $T_u > T_{ps}$, then setting $\alpha = 0$ tends to result in the lowest latency because the user only experiences pull latency. If the user is at the mission site when the mission occurs, i.e. $T_u = 0$, then a setting of $\alpha = 0.5$ results in the best latency because the user will experience the sum of $T_{ps}$ and $T_{pl}$. If the user arrives at the mission site while the data is being pushed, the optimal setting of $\alpha$ depends on $T_{ps} - T_u$ which is then added to $T_{pl}$.

These observations suggest the following guidelines. If the mission is of a type in which personnel typically must travel to the mission site after a mission commences, for example, fire fighters and police responding to a fire, a setting of $\alpha = 0$ is appropriate. If the personnel are already at a mission site when the mission commences, for example guards in a building noticing an alarm, then $\alpha$ should be set to 0.5.

## 2.8 Protocols

In this section we present protocols to implement the algorithms previously discussed. Although we presented the above in terms of centralized algorithms, to implement the algorithms in the context of wireless mesh networks, we prefer a distributed implementation which is scalable and with as little overheads as possible.

We define two distributed protocols that are differentiated by which nodes ex-




(a) Directed by the Source. (b) Directed by the CN.

Figure 2.8: Messing-passage logic of two protocols.

ecute the distributed heuristics. These protocols do not require global knowledge. We assume the sources know the size of all the data items they will push, the location of storage nodes, and have a reasonable approximation of the location of their users. In the first protocol, the data sources run the heuristic and pick the candidate storage nodes. In the second, a node near the mission site called the center node (CN) runs the heuristic and directs the source on where to store its data. The CN may be a node in the center of the expected storage area or the first node to which a source sends a storage request.

### 2.8.1 Directed by the source

A simple protocol is shown in Figure 2.8a. The source runs the heuristic and determines the best candidate node for each of its data items. It then requests that each candidate node store its data, and if they accept the request, pushes the data items. There will be at least as many such requests as there are data items because if the storage nodes are already occupied, the provider will contact its subsequent choices for storage. When the user desires to pull data, it contacts the provider to get an index of where all data is stored and then requests the data it requires from each storage node directly.

Assuming $d_i$ has been assigned to $c_j$, define $O_i$ to be the overhead of this protocol on delivering $d_i$, $C_{ps}$ to be the number of messages between the source and the storage node, $C_{up}$ to be the number of messages between user and source, and $C_{us}$ to be the number of messages between the user and storage node. If we

still use distance as a metric, we have:

$$O_i = \sum_{l \in R_1(d_i)} d(i,l)C_{ps} + \sum_{k \in R_2(d_i)} d(k,j)C_{us} \tag{2.5}$$

where $R_1(d_i) = \{l \mid d_i \text{ requests } s_l\}$ and $R_2(d_i) = \{k \mid u_l \text{ requests } d_i\}$.

Then the total overhead is:

$$\sum_i O_i + \sum_{l \in R_3(u_k)} \sum_k d(k,l)C_{up} \tag{2.6}$$

where $R_3(u_k) = \{l \mid u_k \text{ requests data from } s_l\}$.

### 2.8.2 Directed by the CN

In the general case, storage nodes are likely to be near the users, e.g. when there is more weight on pulling. In this case the source-directed protocol described above will cause request and reply messages from sources to storage nodes to travel many hops. To reduce this effect, we develop a CN-directed protocol in which every source picks a CN as its delegate near the user. Users will retrieve the location index from the CN and pull desired data directly from the nodes (see Figure 2.8b).

Define $C_{pcn}$ to be the number of messages between the provider and its CN. Other constants are similar to the source-directed case. Then the overhead becomes:

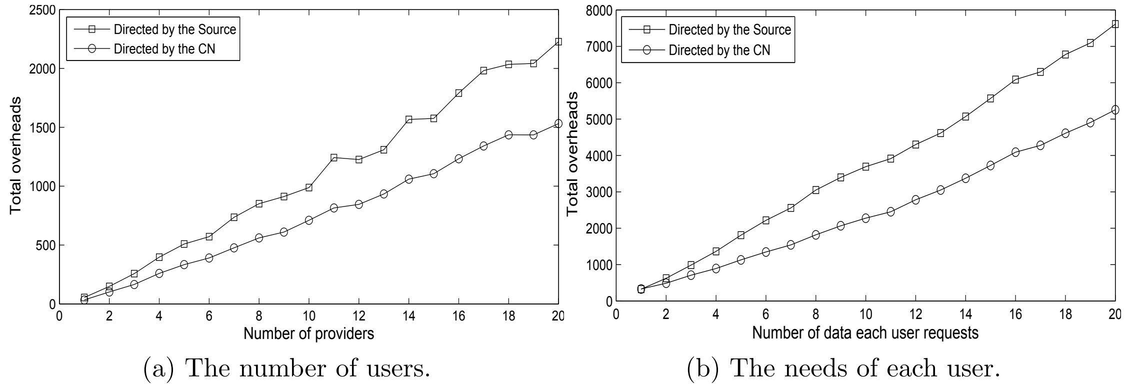
$$O_i = \sum_{l \in R_4(CN_i)} d(i,l)C_{ps} + \sum_{k \in R_5(d_i)} d(k,j)C_{us} \tag{2.7}$$

where $R_4(CN_i) = \{l \mid CN_i \text{ requests } s_l\}$ and $R_5(d_i) = \{k \mid u_k \text{ requests } d_i\}$.

The total overhead becomes:

$$\sum_i O_i + \sum_k \sum_{l \in R_6(u_k)} d(k,l)C_{up} + \sum_m d(s_m, CN_m)C_{pcn} \tag{2.8}$$

where $R_6(u_k) = \{l \mid u_k \text{ requests data from } CN_l\}$.




(a) The number of users. (b) The needs of each user.

Figure 2.9: Amount of overhead, as the indicated parameter varies.

### 2.8.3 Overhead Simulation

To see how the overhead values depend on the problem instance, we conduct two experiments: increasing the number of users and increasing the number of requests from each user. In the tests, all storage nodes have a uniform capacity of 10 units and data items are randomly generated with sizes between 1 and 10 units. Each user requests a certain number of data items randomly from all the data items.

In the first test (see Figure 2.9a), we increase the number of users with each user having a fixed number of requests. The total overheads increase nearly linearly. In the second test, we increase the number of requests from each user while holding constant the number of sources, users and data items. As we can see from Figure 2.9b, the overheads also increase linearly. When each user only requests one data item, the source-directed protocol has slightly less overhead than the CN-directed protocol.

We conclude that as the number of requests rises, either by increasing the number of users or the number of requests per user, the overhead increases linearly. In both of the two tests, the source-directed protocol is outperformed by the CN-directed protocol, again as expected.

## 2.9 Conclusion

To disseminate data to mission sites for efficient access, two latency factors are taken into consideration: hop distance and congestion. In this chapter we devel-

oped heuristics to minimize hop distances and use a virtual occupation technique to disperse data and thereby reduce congestion. We find that our heuristics achieve good performance in a distributed way and that latency can be reduced effectively. We also discussed how a trade-off between push and pull cost/latency can be made, so that in various kinds of situations, users may consistently experience low data-retrieval latency. Finally, we define distributed protocols to implement these algorithms and show that they exhibit acceptable overheads even as the problem size grows.

# Chapter 3

# Time Slot Assignment to Mobile Users

In this chapter, information is disseminated more proactively. Knowing the paths of mobile users and their travel schedules, requested information is downloaded by users as they are moving. The problem is cast as a parallel-machine scheduling problem, with the little-studied property that jobs may have *machine-dependent* time windows.

## 3.1 Introduction

In mobile data access applications [30], mobile clients equipped with wireless communication capability travel within a region, interacting with service providers, i.e., access points or base stations (BS), in order to access information or receive service (see Figure 3.1). For example, in Vehicle-roadside data access [31, 32, 33], pre-deployed RoadSide Units (RSU) provide data access to vehicles as they pass by.

In the mission-oriented scenario we consider, mobile clients travel towards destinations where they have missions to complete, according to precise instructions. For example, in a search and rescue (SAR) mission prompted by an event such as an earthquake or hurricane, the SAR team may need to receive detailed instructions and other information, including e.g. expected numbers of survivors, by the time they reach the location of the emergency, in order to most effectively perform

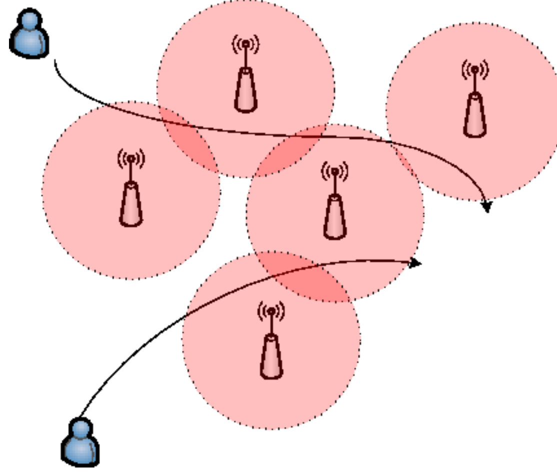


Figure 3.1: Scenario: clients encountering BSs.

their duties.

While certain aspects of these missions may be predictable in advance–certain events will require the services of, say, firetrucks, police, and rescue helicopters–specific details, including the number of responders needed, their initial locations, and the event location–may depend on circumstance. Due to the exigent nature of such events, we may not be aware of all team members' locations until the moment when the mission is initiated; in general, many may be out of BS coverage range. At the same time, with real-time information such as road conditions or the temperature within a burning building, data freshness is crucial. Ideally, we might prefer to deliver just-in-time data at the last possible moment. (In a crowded system, delivering multiple data iterations may not be feasible.) In this motivating scenario, it will thus be necessary to deliver data items wirelessly to team members en-route, as they pass within range of base stations, prior to reaching their destinations.

Since the wireless link has limited range, a client must download its data item from one of the BSs lying sufficiently near to its route. The time to transfer the data depends on the data item size and the channel's transmission rate. Because clients are moving, there will be a limited time window (possibly empty) in which they are able to receive data from each particular BS. In order for a given client to succeed, therefore, its data item must be downloaded from a BS during the feasible time window in which the client is in the communication range. Although more

than one BSs are deployed, many clients may compete for the exclusive usage of a BS's channel. Due to limited bandwidth between BSs and clients, we need to make decision on how to allocate channels to multiple clients and schedule their downloads.

The data items' relative importance is indicated by their *weights*, which may change over time. For example, knowing the road conditions is more important than temperature to the medical team driving to the mission site. However, this importance may decline over time: once the team is near the site, the survivors' medical records may be much more relevant. Since there may be many conflicting data items of various weights, our objective in the problem we study is to maximize the total weight of delivered data, consistent with bandwidth constraints.

This problem can be recast as a parallel-machine scheduling problem. In this interpretation, the wireless links of BSs play the role of machines (with multiple, co-located machines corresponding to the channels of a BS), and the downloading of data items are the jobs. The priority or weight of a data item is interpreted as the *weight* of the job. The goal is throughput maximization, i.e., maximizing the weight of data items delivered. This problem lies within a large family of scheduling problems in which $n$ jobs (each with weight $w_j$ and processing time $p_j$, and release time $r_j$ and deadline $d_j$) are to be assigned to $m$ parallel machines [11]. In some existing settings, job processing time and weight are *machine-dependent*, i.e., their values depend on the machine to which the job is assigned.

However, a crucial aspect of our problem is that BSs are deployed throughout the region so the moment when clients move into or out of the communication ranges differs from BSs to BSs. This makes jobs' release times and deadlines *machine-dependent* in the corresponding scheduling problem. This generalization has received little attention in the past but is an essential aspect of many mobility applications. Mobile services available to moving clients at multiple locations are applicable for this case.

Although the problem without machine-dependent time windows is already NP-hard [11], it is well studied and has some known approximation algorithms. We will cover these algorithms in the Related Work section. In this chapter, we adapt existing algorithms for related scheduling problems and also propose new algorithms, for multiple machine-dependent times settings including offline and

online, which we then evaluate with synthetic data sets as well as real-world data sets obtained from UMass [34].

In addition to the general problems, we consider a realistic environment in which the mobile clients' speed and the amount of available bandwidth at a BS both vary over time, the latter variation due perhaps to unforeseen requests for data. We adapt these algorithms using an estimated bandwidth and show the resilience of different algorithms.

We evaluate the algorithms on cases with uniform item weights and firm deadlines as well as in more general settings in which weights degrade over time, corresponding to soft deadlines. Assuming the weight of a job remains constant over its time window on each machine, we interpret the latter situation into the case with machine-dependent weights. We evaluate the algorithms with different weight degradation functions and summarize the limitations of these algorithms.

The rest of this chapter is organized as follows. Section 3.2 presents related work and Section 3.3 introduces two system architectures. Section 3.4 formally defines the problem and presents the algorithms we adopt and propose, which are then evaluated in Section 3.5. Section 3.6 explores the case in which data item weights are machine-dependent. Finally, Section 3.7 concludes the chapter.

## 3.2 Related Work

This work concerns mobile data access [30] and specifically, vehicle-roadside data access. Systems such as MobiEyes [35] and Thedu [36] have been developed in order to either collect data from or deliver data to moving vehicles. Like in our problem, vehicles can only download data within a time window which is decided by transmission range, distance to the RSU, and the vehicle's speed. Multiple vehicles also compete for the exclusive usage of the common broadcast channel. Therefore, scheduling access by mulitple vehicles within time constraints is the challange. Jiang et al. [37] studied the case in which data are pushed to vehicles periodically; Xu et al. [38] investigated the case in which data are only pushed on-demand, as in our problem. The fundamental difference in the case of our problem is that in general there are multiple BSs at various locations available to the clients. If it fails to download its data at one BS, the client may still succeed

at another BS.

A Content Distribution Network [2] consists of a number of distributed servers whose job is to reduce traffic from data origin servers by delivering content to nearby users. A BS in our problem plays a similar role, viz., delivering information to passing clients. In contrast, however, data items for us are jobs to be scheduled, whereas in a CDN data items are cached in nodes indefinitely.

There is a very large literature on algorithms for scheduling jobs on parallel machines. See [11] for an introduction. Here we refer to some of the most relevant existing work. Lee et al. [39] is the primary antecedent to our work. Their problem, the unrelated machines scheduling problem (USP), minimizes the total weighted flow time, subject to time-window job availability and machine downtime constraints. In USP, job sizes, as well as release times and deadlines, are machine-dependent. Deciding whether it is possible to schedule all jobs is strongly NP-complete [40], even for the case of a single machine and even if only two integer values exist for release times and deadlines [40]. Algorithms are given [39], however, for constrained settings, as well as a zero-one integer programming (IP) formulation, which is solved with branch-and-bound techniques. Lee et al. [39] claim to be the first to study algorithmically scheduling problems with machine-dependent release times and deadlines. We are aware of very little additional work done in the interim. One paper on parallel-machine scheduling [41] considers release times and deadline times to be both job- and machine-dependent. The authors give heuristics only, based on a constraint programming/tabu search hybrid.

Some of the algorithms we implement are drawn from work on scheduling on identical or unrelated machines. Bar-Noy et al. [42] studies several such settings, obtaining algorithms with the following guarantees: a 2-approximation for unrelated machines and a $\frac{(1+1/m)^m}{(1+1/m)^m-1)}$ (which approaches $e/(e-1)$ as the number of machines $m \to \infty$) for identical machines. The algorithm in both cases is called $m$-Greedy (in our notation), which applies an order-by-end-time greedy algorithm machine-by-machine. This algorithm can also be applied to the machine-dependent times setting.

A stack-based Two Phase algorithm with somewhat better performance is given by Berman and DasGupta [43]. Similar algorithms are given by Bar-Noy et al. [42]. These problems reduce job scheduling problems, with sizes, release times and

deadlines, to the *interval scheduling problem*, by replacing a job with all possible (discrete) intervals in which the job can be scheduled. Such intervals are then referred to as *job instances*.

## 3.3 Problem Setting and System Architecture

In this section we first describe information required to solve the problem. Then we discuss three problem settings and present two system architectures on which we base our solutions.

In the system we have base stations deployed within a geographical region and mobile clients with information needs that are traveling towards a mission site. We call the time period during which a client can talk to the BS *contact time window*. For example, in Figure 3.2, the client enters communication range of the BS at time $t_1$ and leaves at $t_2$, so the contact time window duration is $t_2 - t_1$. A client must retrieve its information from any one of the BS within this time window. The window is decided by the speed vector, the route and the relative location of the BS and the client. Thus, each BS/client pair may have a unique contact time window.

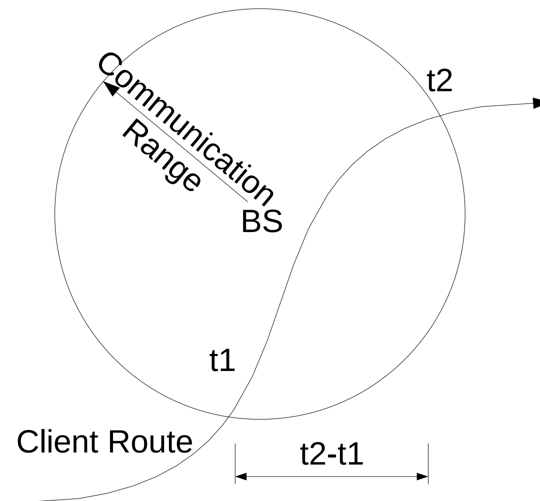



Figure 3.2: Contact Time Windows

Each data item (or *job*) has a size, and each client has a transmission rate with each BS. The time required to complete the transmission of a data item depends on

these two values. Again, this time varies by BS/client pair, and may even change over time. We call this duration the *download time*. As well as a size, each data item has a weight value.

The system has slotted time, so the task of the system is to decide how to allocate these timeslots to different clients knowing all or part of the following information: contact time window, download time and weight for each BS/client pair.

### 3.3.1 Problem Settings

Based on how much information we have when solving the problem, we define three types of problem settings:

1. Offline
2. Centralized Online
3. Distributed Online

In the offline setting, everything required to solve the problem is known in advance, which typically cannot be assumed. Studying the offline problem is nonetheless useful, however, since the optimal offline solution value serves as a baseline in evaluating online performance. Similarly, an offline algorithm with an approximation guarantee can be used to provide looser baseline comparison.

In the centralized online setting, a centralized server collects information and schedules the downloading based on its current knowledge. The server server is unaware of each client until it appears. For example, in an SAR mission, a command center controls the delivery of information to the team members, but not until the beginning of the mission does it know about which members are needed and their current locations. Even after the mission starts, it does not know anything about the future needs. Therefore, it can only schedule the downloading according to its current knowledge and make adjustment later on. The results at each step are treated as reservations which may or may not be preempted by new coming clients.

In the distributed online setting there is no centralized server with knowledge of all clients' plans. Every BS is responsible for allocating its own bandwidth;

no advanced reservations are allowed. Missions arrive spontaneously, and paths are planned dynamically. For example, a handheld device user may suddenly request to download information. The client then sends a request to a BS within communication range, including data description and weight. Upon receipt of such a request, the BS can calculate its time window based its communication range and the client's speed vector. Clients whose requests are rejected may then turn to other BSs.

### 3.3.2 System Architectures

Given the possible problem settings, we propose two system architectures for generating the delivery schedules as described below.

In the centralized system a single server receives information updates from all locations and acts as the scheduler. This server is assumed to be powerful, able for example to solve IP (integer programming) problems. We use this server to solve the problem near-optimally by IP or offline algorithms. However, in practice unpredictable things may happen which degrade performance. For example, bandwidth may change, a complication that is difficult to integrate into the IP or offline algorithms. Or in another case, new jobs appear from time to time, so frequent information updates will require that the centralized scheduling algorithms be run frequently, which may involve unacceptable system overhead.

In the distributed system, we trade optimality for flexibility. On the one hand, (near-) optimal solutions will be harder to obtain in the absence of global information. On the other, a distributed system is more suitable for highly dynamic problems because machines can make scheduling decisions based on local and current information. Moreover, with each node using best-effort algorithms, such computations will be much faster and involve lower overhead.

Although online distributed algorithms typically will perform less well than centralized algorithms, we will see in Section 3.5 that their performance can be quite good.

The relative merits of these systems are summarized in Table 3.1. We will further discuss this when we introduce the algorithms.

| | Centralized | Distributed |
|---|---|---|
| Highly dynamic problem | Bad | Good |
| Overhead | High | Low |
| Solution | Near-Optimal | Good |

Table 3.1: Comparison of System Architectures

## 3.4 Problem Model and Algorithms

In this section we provide a formal problem definition and define and compare two Integer Programs (IPs) to solve the problem. We then discuss approximation algorithms for the general case of the problems followed by presenting optimal solutions for some special cases.

### 3.4.1 Problem Models

In our model, we interpret the downloading of data as *jobs* and communication channels of BSs as *machines*. Each job corresponds without loss of generality to a client traversing a route, seeking to obtain one data item. A client desiring multiple data items is assumed to be represented either by a client seeking a single complex item or as multiple clients seeking individual items while traveling the same route.

Some BSs may have multiple channels to communicate with clients. Then the channels of a BS can be construed as several co-located machines. Each machine can communicate with at most one job at a time. For practical reasons, we do not allow jobs to be paused and continued, or to switch channels; jobs can only be preempted and start again from the beginning.

Given are jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. The clients roaming through the geographical region come within transmission range of some of the machines at different times. This gives rise to release times and deadlines of jobs that are both job- and machine-dependent. We use indices $j, i \in \{1, ..., n\}$ for jobs, $k \in \{1, ..., m\}$ for machines, and $s \in \{1, ..., t\}$ for timeslots.

Let $r_{jk}$ and $d_{jk}$ be the release time and deadline, respectively, for job $j$ from the point of view of machine $k$. $w_{jk}$ is the weight of job $j$ on machine $k$. We use the flexibility of machine-dependent weights in order to implement "soft" deadlines,

in which the value of a data item declines over time. Since each client traverses a known path, at known speeds, from source to destination, machine location can be used as a proxy for the passage of time. More precisely, this $w_{jk}$ could be modeled as $w_{jt}$ because the weight varies over time. For simplicity, however, we assume this weight stays constant within a given machine's contact time window, and so it suffices to parameterize $w$ by $j, k$.

$p_{jk}$ is the processing (or download) time for client $j$ on machine $k$, which, in some settings, may be based on a job size and a machine speed. Let $S_{jk}$ indicate the starting time of job $j$ on machine $k$, if this assignment is chosen. In this case, we must have $r_{jk} \leq S_{jk}$ and $S_{jk} + p_{jk} \leq d_{jk}$. A *job instance* is not a job but a potential assignment of a job, i.e., a feasible interval of size exactly $p_{jk}$ on some machine $k$ in which job $j$ could be run. As noted above, a scheduling problem can be construed as an interval selection problem by replacing a job/machine pair, and its release time, deadline, and processing time, with the set of all possible corresponding job instances. Some of the algorithms, including the Two Phase [43], operate by considering *job instances* in order of increasing end time. The notations used in this chapter are summarized in Table 3.2.

| **Term** | **Definition** | **Term** | **Definition** |
|---|---|---|---|
| $J_j$ | Job $j$ | $M_k$ | Machine $k$ |
| $r_{jk}$ | $J_j$'s release time on $M_k$ | $d_{jk}$ | $J_j$'s deadline on $M_k$ |
| $p_{jk}$ | $J_j$'s process time on $M_k$ | $w_{jk}$ | $J_j$'s weight on $M_k$ |
| $S_{jk}$ | $J_j$'s start time on $M_k$ | $t$ | Number of timeslots |
| $n$ | Number of jobs | $m$ | Number of machines |

Table 3.2: Table of Notations

### 3.4.2 IP Formulations

Our problem can be expressed within the formalism of integer programming (IP). Although solving IPs is NP-hard in general, it is frequently possible in reasonable time for moderately sized instances. For larger instances, solving a linear programming (LP) relaxation in polynomial time can also provide useful upper bounds on solution quality.

In this section, we present two different IP formulations for the problem. We found in our experiments that which one is faster to solve depends on the nature of the problem instance, about which we elaborate below.

#### 3.4.2.1 Start-time Formulation

We extend the single-machine formulation of [44] to multiple machines, by introducing an additional summation index $k$. Following [44], we define the following variables:

$$y_{jk} = \begin{cases} 1 & \text{if job } j \text{ is performed on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1 & \text{if job } i \text{ is performed before job } j \text{ on} \\ & \text{machine } k\text{, or job } j \text{ is performed} \\ & \text{on machine } k \text{ and job } i \text{ is not,} \\ 0 & \text{otherwise} \end{cases}$$

$$S_{jk} = \text{the start of job } j \text{ if it is performed on machine } k$$

The Start-time Formulation is shown in Table 3.3. The first two lines of constraints implement the time windows; the constraints of lines 3 through 6 provide for mutual exclusion as follows. If jobs $i, j$ are both assigned to machine $k$, then lines 5 and 6 force a decision as to the scheduling order of these jobs. Given this, lines 3 and 4 then ensure that jobs $i, j$ do not overlap. Note that because $t$ upper-bounds all deadlines, and one of the two variables $x_{ijk}, x_{jik}$ in this case equals 1, one of these two constraints is satisfied trivially.

#### 3.4.2.2 Time-indexed Formulation

Time-indexed formulations (see Dyer [45]) involve 0/1 decision variables for each possible assignment. This formulation involves only one set of decision variables, but they have three indices: job, machine, and timestep.

We define a variable $x_{jks}$ for each job instance $[s, s + p_{jk})$ of $J_j$ on $M_k$. The objective is again to maximize the sum of weights of scheduled jobs. The Time-indexed Formulation is shown in Table 3.4.

The first set of constraints prevents multiple jobs from being scheduled si-

$$
\begin{aligned}
\max \ & \sum_{j=1}^{n}\sum_{k=1}^{m} w_{jk} \cdot y_{jk} && && (3.1)\\
\text{s.t. } & S_{jk} \geq r_{jk} y_{jk} && \forall j,k\\
& S_{jk} \leq (d_{jk} - p_{jk}) y_{jk} && \forall j,k\\
& S_{ik} - S_{jk} \geq p_{jk} y_{jk} - 2t x_{ijk} && i \in [1, n-1], j > i, \forall k\\
& S_{jk} - S_{ik} \geq p_{ik} y_{ik} - 2t x_{jik} && i \in [1, n-1], j > i, \forall k\\
& x_{ijk} + x_{jik} \geq y_{ik} + y_{jk} - 1 && i \in [1, n-1], j > i, \forall k\\
& x_{ijk} + x_{jik} \leq 1 && i \in [1, n-1], j > i, \forall k\\
& \sum_{k=1}^{m} y_{jk} \leq 1 && \forall j\\
& x_{ijk} \in \{0,1\} && \forall i \neq j, k\\
& y_{jk} \in \{0,1\} && \forall j,k\\
& S_{jk} \in \{0,1,2,...\} && \forall j,k
\end{aligned}
$$

Table 3.3: Start-time Formulation

$$
\begin{aligned}
\max \ & \sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{s=r_{jk}}^{d_{jk}-p_{jk}} w_{jk} \cdot x_{jks} && && (3.2)\\
\text{s.t. } & \sum_{j=1}^{n}\sum_{u=s-p_{jk}+1}^{s} x_{jku} \leq 1 && \forall k,s\\
& \sum_{k=1}^{m}\sum_{s=r_{jk}}^{d_{jk}-p_{jk}} x_{jks} \leq 1 && \forall j\\
& x_{jks} \in \{0,1\}
\end{aligned}
$$

Table 3.4: Time-indexed Formulation

multaneously on any single machine; the second set prevents any job from being scheduled more than once.

#### 3.4.2.3 Comparison of the Formulations

First we list the number of variables and constraints in Table 3.5. In the Time-indexed formulation, the program size grows linearly with all three parameters (the number of jobs $n$, the number of machines $m$, the number of timeslots $t$). In the Start-time formulation, the program size is independent of timespan $t$; instead, release times and deadlines (bounded by the timespan $t$) appear as constants. The program size grows quadratically with the number of jobs $n$, however.

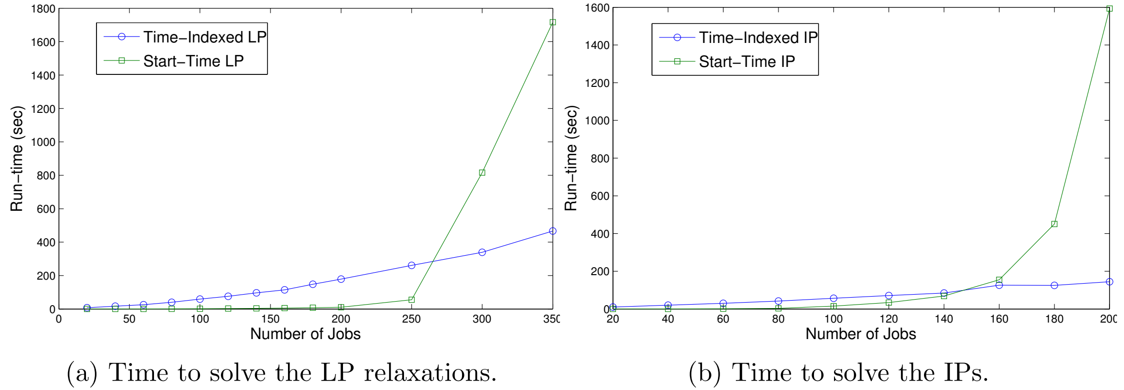
| | Start-time | Time-indexed |
|---|---|---|
| Variables | $O(n^2m)$ | $O(nmt)$ |
| Constraints | $O(n^2m)$ | $O(\max(n, mt))$ |

Table 3.5: Complexity of Two Formulations

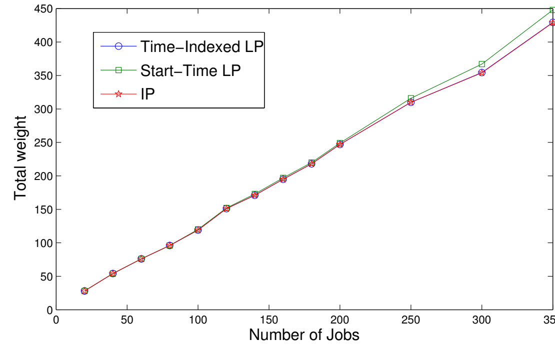
To compare the actual running speed of the two formulations, we performed the following test: the number of machines is fixed at 25 and the total running time set to 1000 while we increase the number of jobs. All jobs are randomly generated. We implement the programs in AMPL [46] and solve them with CPLEX [47], measuring the computation times. Note that we turn the presolve option in AMPL off, so no variables nor constraints get eliminated in AMPL and the CPLEX solver solves the problem. Turning on presolve (omitted here) produces similar results.

Figures 3.3a and 3.3b compare the solver's running times for the two LP-relaxations and two IPs, respectively. For small instances of the problem, the Start-time LP formulation is solved faster, as expected. For larger instances, though, solving Time-indexed LP is more efficient. The two LP-relaxations give different upper-bounds (see Figure 3.3c), however. The time-indexed formulation gives tighter bounds. This difference can be significant for larger problem instances. In fact, [48] claims that for many IP problems, LP-relaxed time-indexed formulations can provide tighter bounds than other sorts of formulations. The tighter bounds produced by a solution of time-indexed formulation lead to more robust branch-and-cut algorithms. Figure 3.3c shows that the upper-bound for the Time-indexed LP almost coincides with the solution to the IP. The Start-time LP gives a higher upper-bound.

As a result, in practice we prefer using the Time-indexed formulation in our experiments because the running time increases linearly as we increase the problem




(a) Time to solve the LP relaxations. (b) Time to solve the IPs.




(c) Optimal values of the LPs and the IP.

Figure 3.3: Two Formulations

size. In cases in which the Start-time formulation is faster, i.e., when $n^2m \ll nmt$, we choose it.

### 3.4.3 Algorithms and Techniques for General Cases

The general problem is NP-hard, so first we discuss approximation algorithms in general cases.

#### 3.4.3.1 Offline Algorithm

In offline settings, we can try to solve the IP problem. Although this may take too long for realistic problem instances, we can use these solutions to evaluate other algorithms. In fact, we provide solutions to the LP as a bound on the optimal solution. In practice, we typically find the LP and IP optimal solution values to be comparable.

We use the Two Phase algorithm from [43] as one of our offline algorithms. In the first phase, it pushes job instances in order of non-decreasing right endings onto a stack, if they have great enough weight relative to conflicting jobs on the stack; in the second phase, it pops job instances from the stack and places them in a nonoverlapping schedule. When a job enters the stack, it is pushed with the (strictly positive) difference of its weight and the weight of the overlapping jobs lower on the stack, with the effect that the total weight of the stack equals the weight of the schedule formed in the second phase. This algorithm provides a 2-approximation guarantee. See Algorithm 5 for details of this algorithm.

**Algorithm 5** Two Phase Algorithm

1: $L \leftarrow$ the set of all job instances *(job, weight, beginning, ending)*
2: sort $L$ so the ending is non-decreasing
3: $S \leftarrow$ an empty stack
4: **for** each $(i, w, d, e)$ from $L$ **do**
5: $\quad v \leftarrow w - total(i, d) - TOTAL(d)$
6: $\quad$ **if** $v > 0$ **then**
7: $\quad\quad$ push$((i, v, d, e), S)$
8: $\quad$ **end if**
9: **end for**
10: **for** each $i$ **do**
11: $\quad done[i] \leftarrow$ false
12: **end for**
13: $occupied \leftarrow t$
14: **while** $S$ is not empty **do**
15: $\quad (i, v, d, e) \leftarrow$ pop$(S)$
16: $\quad$ **if** $done[i] =$ false and $e \leq occupied$ **then**
17: $\quad\quad$ add $(i, d, e)$ to solution
18: $\quad\quad done[i] \leftarrow$ true, $occupied \leftarrow d$
19: $\quad$ **end if**
20: **end while**

Another combinatorial algorithm called Admission (see [42]) provides a $3 + 2\sqrt{2}$-approximation. In this algorithm, jobs are considered in the order of non-decreasing endings on one machine. If the currently considered job overlaps with a set of scheduled jobs, we compare this job's weight with the sum of the overlapping jobs' and decide whether to replace them with the current job. To apply this to

$m$ machines, we call the Admission algorithm $m$ times ($m$-Admission), machine by machine. The $3 + 2\sqrt{2}$ ratio still holds in this situation [42]. We evaluate this algorithm as well as the Two Phase algorithm because it may be extended to work in the centralized online case.

#### 3.4.3.2 Centralized Online Algorithm

As mentioned above, Admission works in real time order, so other than applying it machine by machine, it is easy to extend the algorithm to work on machines in parallel. We call the extended algorithm Global Admission (see Algorithm 6), in which we schedule the earliest finishing job among all the machines at each step.

**Algorithm 6** Global Admission

```
1: A ← ∅
2: I ← the set of all job instances
3: while I is not empty do
4:    let J_j ∈ I be a job instance that terminates earliest
5:    I ← I \ {J_j}
6:    let C_j be the set of jobs in A overlapping with J_j
7:    let W be the total weight of C_j
8:    if W = 0 or w_jk > W · (1 + ℓ_j/L_j) then
9:       A ← A ∪ {J_j}\C_j
10:   end if
11: end while
12: return A
```

The following criterion is used in $m$-Admission to decide whether a job should replace existing jobs it conflicts with. Let $W$ be the total weight of all scheduled jobs overlapping with the current job $j$. We accept job $j$ if $w_{jk} > W \cdot \beta$, for some constant $\beta$ ($\beta > 1$ prevents repeated preemptions and guarantees a constant approximation factor; a typical $\beta$ value is $1+\sqrt{2}$ that minimizes the approximation factor).

The motivation to modify this criterion is illustrated in Figure 3.4. Two situations are shown, both with overlapping job instances of size 2 and 4. The job instance of size 2 is considered first in both cases because it ends first and we take

jobs in order of increasing end time. If it is canceled for the job instance of size 4, the non-overlapping portion (if any) of its timeslots is wasted. So in case $a$ one timeslot is wasted if we cancel the first job instance, while in case $b$ both timeslots of the first job are reused. As a result, we should expect higher weight on the second job in case $a$ than in case $b$ in order to replace the first job. However, a constant $\beta$ does not differentiate between the two cases.

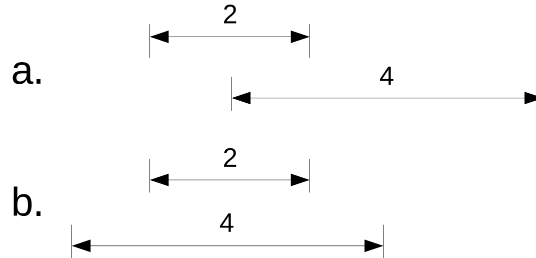



Figure 3.4: Two cases of conflicting jobs.

More generally, if the new job $j$ conflicts with a set of jobs $C_j$, let $\ell_j$ be the distance between the right endpoints of job $j$ and the rightmost job in $C_j$. (Note that all jobs conflicting with $j$ must end earlier than it.) And let $L_j$ be the span of $C_j$. We modify the criterion given above so that we accept job $j$ if $w_{jk} > W \cdot (1 + \ell_j/L_j)$. With this criterion, the size-4 job will be less likely to replace the size-2 job in case $a$ than in case $b$. We conducted experiments comparing the two criteria and found that the modified criterion consistently outperforms the previous one. We therefore use this modified criterion in all Admission-related algorithms in our experiments.

We apply Global Admission iteratively to realize our Centralized Online algorithm (see Algorithm 7). We use Global Admission rather than Two Phase because it schedules jobs in real time. Two Phase is not suitable for real-time use because it does not begin scheduling jobs until after iterating through all jobs. The main idea of this algorithm is to reserve timeslots for a job on the machine on its path. The reservation is not guaranteed until the job starts executing on that machine, however. Until then, the reservation is subject to cancelation or modification as new jobs arrive. It is natural to view this as an incremental offline problem in which we run an offline algorithm at the moment when new jobs arrive.

For a highly dynamic system, it may become burdensome to run Global Ad-

**Algorithm 7** Centralized Online

1: **for** each moment $t$ a new job $J_j$ arrives **do**
2: fix all scheduled jobs $J_i$ with $S_{ik} \leq t$
3: remove other jobs from the scheduled job list
4: call Global Admission with all unscheduled jobs
5: **end for**

mission too often.

#### 3.4.3.3 Distributed Online Algorithm

In the Distributed Online setting, no reservations are allowed; each job requests timeslots from the machines within its communication range. Each time a machine receives a new job request $J_j$, it adds the job to its candidate list $I$ and record its $p_j$, $w_j$ and $d_j$ (we assume $d_j$ is known as mentioned in Section 3.3.1). If no job is currently running, the machine schedules an available job $J_i$ maximizing $w_i/p_i$; if some job $J_r$ is currently running, we kill $J_r$ and schedule $J_i$ with the largest $o_i = w_i - w_r \cdot (1 + \frac{\ell_i}{p_r})$ (if this $o_i$ is positive). Once a job is scheduled, it cancels its requests to other machines; once a job is killed, it can again request other machines and start from the beginning. There might be cases with repeated job preemption and only the last job getting scheduled. But the criterion makes sure that the last job has a much larger weight than the sum of the cancelled ones so that the total scheduled job weight is still comparable to the optimal. See Algorithm 8 for details.

### 3.4.4 Optimal Algorithms for Special Cases

The crucial aspect of this scheduling problem is the machine-dependent time windows. In general, the problem is NP-hard even without machine-dependent release times and deadlines, so the best we can hope for is approximation algorithms. In some special cases, however, our problem can be solved optimally.

**Algorithm 8** Distributed Online

```
1: // occupied will be the last occupied timeslot
2: // I will be the set of unscheduled jobs
3: occupied ← 0
4: I ← ∅
5: for each moment t do
6:    given incoming job J_j, I ← I ∪ {J_j}
7:    if occupied ≤ t then
8:       schedule J_i ∈ I which has the highest w_i/p_i
9:       I ← I\{J_i}
10:      occupied ← t + p_i
11:   else
12:      running job J_r
13:      J_i ∈ I is job with the largest o_i = w_i − w_r · (1 + ℓ_i/p_r)
14:      if o_i > 0 then
15:         replace job J_r with J_i
16:         I ← I ∪ {J_r}\{J_i}
17:         occupied ← t + p_i
18:      end if
19:   end if
20:   remove from I any jobs no longer schedulable
21: end for
```

#### 3.4.4.1 Unit Processing Time

If jobs have unit processing time, the problem reduces to a maximum matching problem in which jobs are matched to timeslots. This matching problem can be solved optimally in polynomial time by classical algorithms such as the Hungarian algorithm.

In one even more special case, when all release times are zero ($r_{jk} = 0$) and each machine encounters the jobs in the same order, the problem can be solved in $O(n \log n)$ time. We refer to this type of monotonicity as **mono1**[1]. For this setting, we can adopt the optimal algorithm for the corresponding setting without machine-dependent windows [49]. In that algorithm, we sort jobs by deadline (note that this ordering is machine-independent), and then iteratively schedule

[1]Below, we consider **mono2**, monotonicity from the point of view of the jobs. Note that these are two separate notions, which can occur separately or together.

jobs in the earliest available slot. If there is no available slot for the job under consideration, then that job replaces an already scheduled job of minimum weight, if doing so would increase the schedule weight; otherwise, the job is discarded.

#### 3.4.4.2 Arbitrary Processing Time with mono1

This problem, as well as the generalization with arbitrary but equal processing times on all machines, is solvable in pseudo-polynomial time by the dynamic programming (DP) techniques of Rothkopf [50] and Lawler and Moore [51]. In the machine-dependent deadlines / release time zero case, different machines encounter jobs in the same order, and so we can directly apply the DP algorithms. Unfortunately, the running time and space are both $O(n(\sum p_j)^m)$, which becomes impractical for even moderate values of $m$. Neither of these algorithms applies to the setting with varying release times, however.

## 3.5 Performance Evaluation

In this section, we evaluate the algorithms for general cases, using both synthetic and real-world data. Then we test them in a more realistic case in which network bandwidth changes. We use uniform item weights and firm deadlines in this section; that is, $w_{jk} = w_j$. The cases with weights degrading over time and soft deadlines are tested in Section 3.6.

### 3.5.1 Mobility Pattern and Data Generation

It takes two steps to generate a problem instance. The first step is to generate the mobility pattern of the clients, thus generating the time windows. The second step is to generate data descriptions, including data sizes and weights to calculate the download times with a proper bandwidth value.

In our experiments, the movement of the clients follows one of two mobility patterns. In the first set of experiments, we use synthetic traces, randomly generated using three movement patterns and Random Waypoint [52]. The BSs are deployed in a grid. In the second set of experiments, we extract mobility patterns

from real mobility traces (obtained from UMass [34]) of buses encountering access points (APs) while following their routes.

In our tests, as mentioned in Section 3.3, there is one data item per client. Their sizes are uniformly distributed within $[1, 50]$ units. The default bandwidth is 1 unit per timeslot. The time horizon $t$ is set to be 1000 timeslots. Data weights are chosen from a Zipf distribution, clipped with a minimum weight of 1 and a maximum weight of 10.

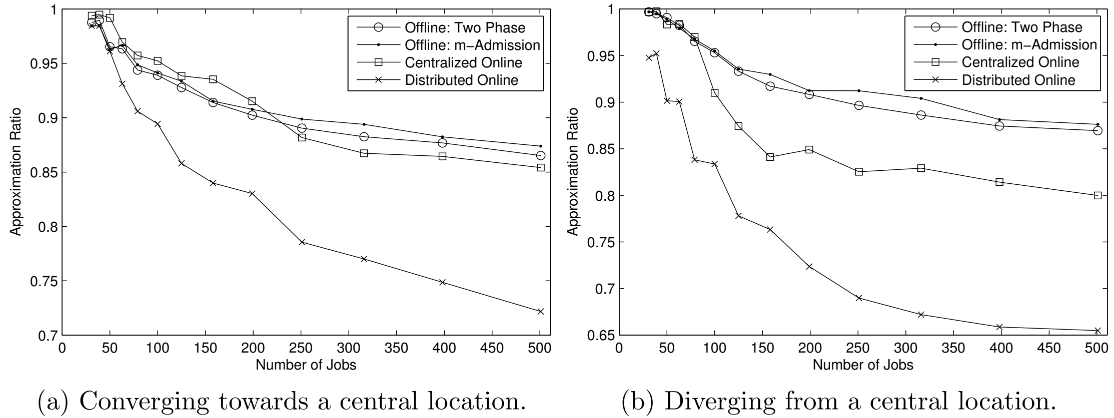
### 3.5.2 Simulation with synthetic mobility patterns

In the first series of simulations, we generate three different movement patterns in order to simulate three possible scenarios:

1. all clients converging towards a central location from disparate locations
2. all clients diverging from a central location to disparate locations
3. all clients traversing the same route, encountering the machines in the same order (**mono2**)

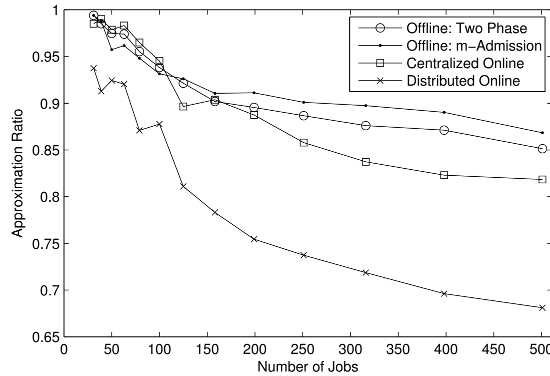
For any of these three patterns, we randomly pick the starting location or the destination of a client and its speed. We fix the number of machines (BSs) to 25 and vary the number of jobs (data items). For each job count, we randomly generate 10 different problem instances and run the algorithms. We also solve the LP relaxation, which provides an upper bound on the optimal solution value. We divide all other results by this upper bound, so the value shown is a lower bound of the approximation ratio.

Although problem difficulty varies *between* these three scenarios, within each setting we generally see similar patterns among the algorithms' performance (see Figure 3.5). When the system is not busy, all the algorithms achieve close to the optimal. As the problem instance grows, the Distributed Online's solution quality falls the fastest. The Centralized Online algorithm performs only slightly worse than the two offline algorithms (which recall provide constant-factor approximations). Even the Distributed Online algorithm, however, outperformed the offline algorithms' worst-case guarantees, consistently achieving more than 70% of offline optimal value. Because it schedules current job without knowing the later jobs




(a) Converging towards a central location.

(b) Diverging from a central location.




(c) Traversing the same route.

Figure 3.5: Special traffic patterns.

may have higher weight. If new jobs come with higher weight, it will keep on finishing the running jobs.

To test the robustness of the algorithms we conducted another test using the Random Waypoint model. In Random Waypoint model, a node repeatedly selects a random destination in the simulation region and a speed from a range. However, we see very similar results to those in Figure 3.5.

We use the same settings in the second simulation, except this time varying the maximum job size (see Figure 3.6). As the max job size increases, the approximation factors of all algorithms drop at first and then begin rising again. The performance effects of increasing the max job size are complex. Our interpretation of the results is that at first, having larger jobs makes scheduling more difficult for the approximation algorithms. Because each job will occupy more timeslots thus

make the competition more severe. Over time, though, as the maximum job size (and hence the average job size) continues to grow, more and more jobs become infeasible for some machine (because they are too large). Thus, the the competition actually becomes less severe and the relative performance of the approximation algorithms improves.

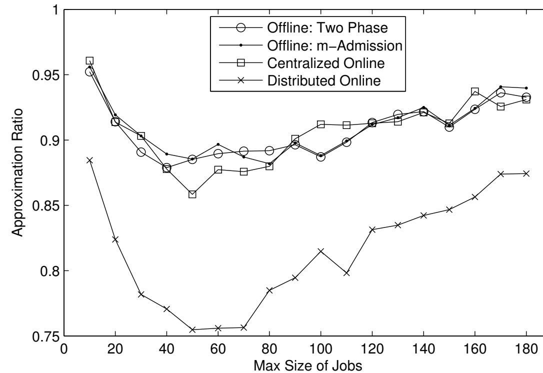



Figure 3.6: Performance comparison, varying maximum job size.

### 3.5.3 Simulation with Trace Files

We analyzed several bus/AP trace files obtained from UMass [34]. While these traces have useful mobility characteristics in terms of contact initiation and duration, there are not enough bus routes to simulate a heavily taxed system. Therefore, we replay several traces simultaneously, treating buses from different trace files as different buses, in order to obtain sufficiently many parallel jobs. We consider the $m$ busiest BSs in these simulations.

A histogram of the window sizes found in this dataset is shown in Figure 3.7. Based on this data, we chose a maximum job size of 50, since larger jobs would not be schedulable anywhere.

We still limit the simulations to one data item per client. The data items are generated randomly as described in the beginning of this section. As we can see (Figure 3.8), the algorithms tested achieve a very good approximation of the optimal. Note that the Centralized Online algorithm knows the arrival of jobs in future time windows.

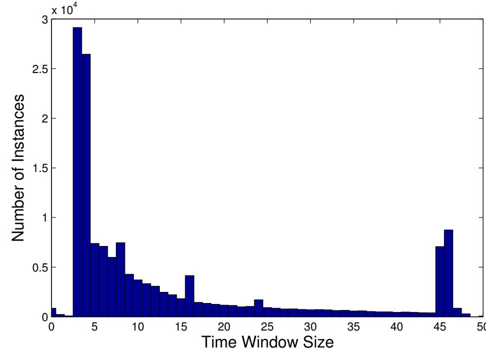



Figure 3.7: Histogram of time window size

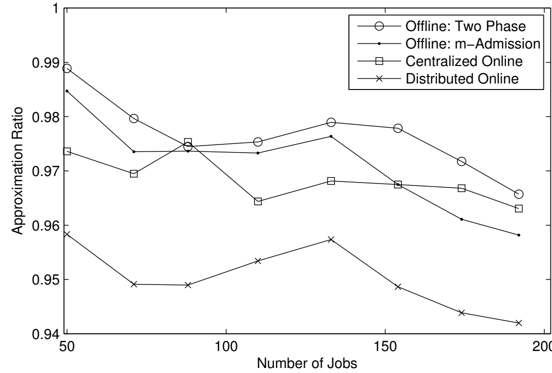



Figure 3.8: Performance comparison, increasing the number of jobs.

### 3.5.4 Varying Bandwidth

In real-world deployments, network conditions can vary over time, so the effective transmission rates at BSs may not be constant. In this case, the processing time $p_{jk}$ depends on time as well as on job and machine. We cannot expect to know in advance about such bandwidth changes; since the offline algorithms make decisions in advance, we need to rely on bandwidth estimates; the Centralized Online algorithm will also use these estimates. Unfortunately, if these estimates are over-confident, then the schedule produced may not actually be feasible; if the estimates are over-pessimistic, then this means wasting bandwidth we could have used. The Distributed Online algorithm uses a real-time rate estimate for current bandwidth, but it may also experience the problem of producing infeasible schedules since bandwidth may change as a job is executed.

To investigate this situation, we conducted the following experiment: the band-

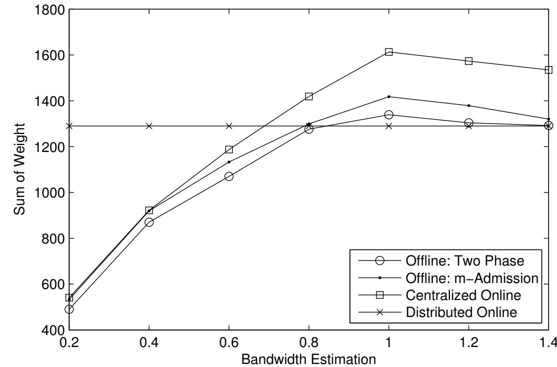



Figure 3.9: Performance comparison, increasing the estimated bandwidth.

width of each machine changes over time, following a normal distribution with expectation 0.5 and variance 0.2, with the distribution clipped at $[0.1, 1]$. As the bandwidth changes, the processing time changes accordingly. We again use the UMass traces to generate time windows, with other parameters set as in the previous tests. In order for our algorithms to achieve good performance, we increase the estimated bandwidth to observe the impact on our solutions to the same problem instance and to find the best bandwidth estimation. Because integrating this generalization into the IP formulations would yield intractably large programs, we do not provide an optimal bound for comparison. The results shown in Figure 3.9 plot the performance of the different algorithms.

From Figure 3.9, we can first of all see that the Distributed Online algorithm achieves the same results because it always makes decision based on the current bandwidth rather than a predefined bandwidth value. When the rates are underestimated, the three other algorithms performed very poorly. This is because we waste time by assigning jobs more timeslots than they need. As the estimated rate increases, the offline and centralized algorithms begin to outperform the Distributed Online algorithm. When we overestimate the rates, however, the total weight drops. Many scheduled jobs do not complete since they require more time than allocated.

Interestingly, the Centralized Online beats $m$-Admission and Two Phase. In the fixed bandwidth situation, we expect the opposite; in the case of varying bandwidth, however, scheduling jobs too tightly has a negative effect. Leaving some extra space between jobs makes the schedule more robust, providing some

leeway to accommodate varying rates. The distributed algorithm naturally fits this situation without any estimation on rate required while still performing well.

## 3.6 Machine-dependent weights

We now generalize data item weights, along with time windows and download times, to vary by machine. The first motivation of machine-depending weights is to implement soft deadlines, in which data item values decline over time, or equivalently when downloaded from more distant BSs. A second and opposite motivation is to privilege fresher data, downloaded as late as possible. In this section we first discuss several weight degradation functions and their meanings in practice. Then we test our algorithms using data item whose values change according these weight degradation functions, in problem settings similar to those in Section 3.5.

### 3.6.1 Degradation Functions

Let $w_j(t) = w_j \cdot f(t)$, where $w_j$ is the initial weight of job $j$ and $f(t) \in [0, 1]$ indicates a *degradation function*. We evaluate three such functions (time $t \in [0, 1]$ is normalized):

1. Linear decreasing: $f(t) = 1 - t$

2. Decreasing by *arccot*: $0.5 - \arctan(10t - 5)/\pi$

3. Linear increasing: $f(t) = t$

Linear decreasing indicates the case in which each data item needs to be delivered immediately; the weight drop is proportional to the lateness. Decreasing by *arccot* (Figure 3.10) is similar to the linear decreasing case, except that the weight decreases mildly first and then suddenly drops to almost zero; this function is related to the case in which data is all useful until a certain time, after which it becomes nearly useless. A linear increasing function may be used when we prefer a *just-in-time* delivery of data, that is, the earlier the data is delivered, the longer the storage resource is occupied.

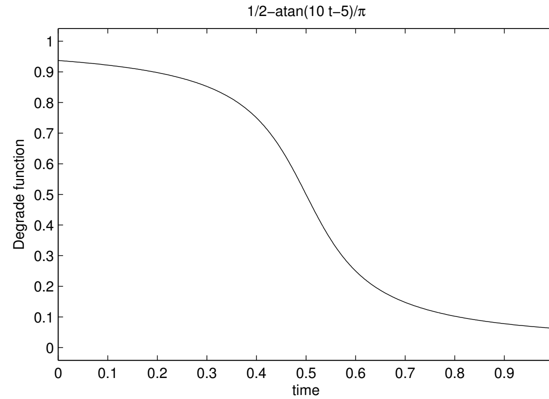
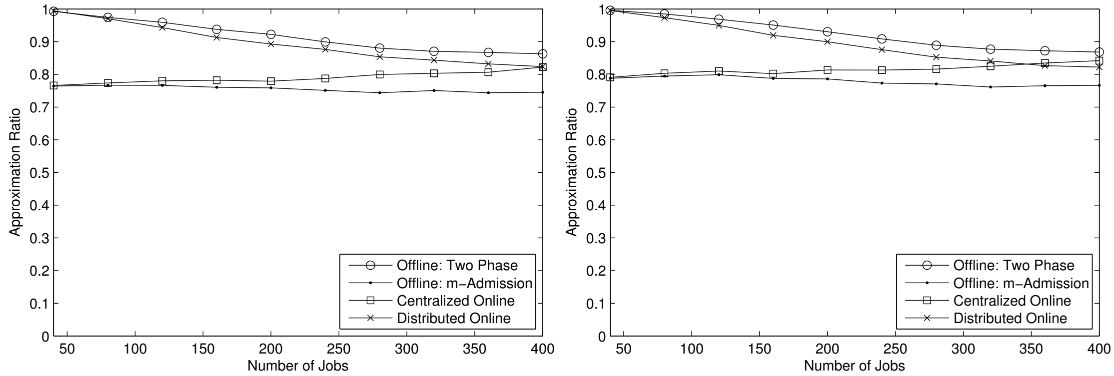



Figure 3.10: Decreasing by *arccot*

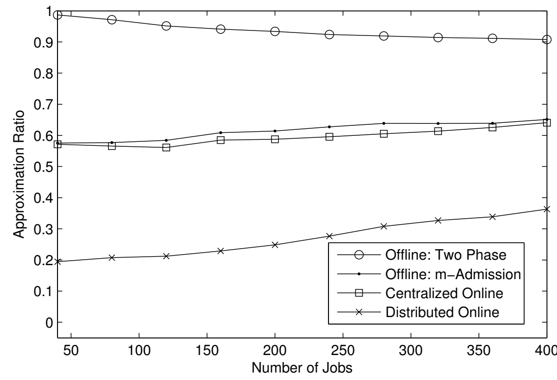
As mentioned in Section 3.4, to make our algorithms compatible, we interpret weights changing over time as machine-dependent weights, assuming a job's weight remains constant over its time window on each machine. For example, job $j$'s weight is $w_j(t)$, as a function of time $t$; the job is available on machine $k$ at time $r_{jk}$ and goes out of range at time $d_{jk}$. We define the weight on machine $k$ to be the value when the job becomes available, that is, $w_{jk} \triangleq w_j(r_{jk})$. We run our algorithms with this machine-dependent weights $w_{jk}$ as an approximation of time-varying weights.

### 3.6.2 Algorithm performance

Although our algorithms are not specifically designed for the machine-dependent weight case, they are not incompatible with it. We performed simulations similar to those described in Section 3.5, in order to learn how the algorithms perform in this more general case. In each test, we increase either the number of jobs or the maximum job size while holding the number of machines fixed. The weight of each job changes according to one or another of the degradation functions presented above. We again compute the LP optimal and present the results in the form of lower bounds on achieved approximation factors. Since there are more parameters involved than in the previous tests (we need a matrix instead of a vector to store the weights), we tested on smaller, easier problem instances so that the LP solver would run in reasonable time.




(a) Decreases linearly. (b) Decreases following inverse cotangent.



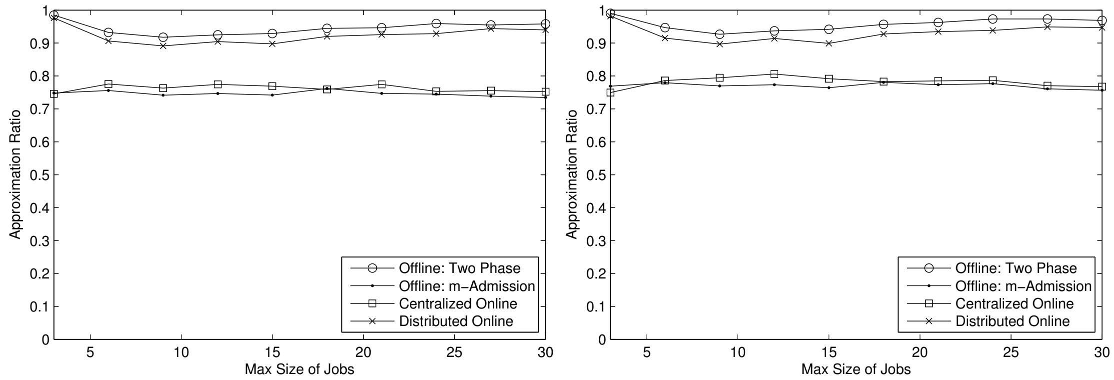

(c) Increases linearly.

Figure 3.11: Test on machine-dependent weights, varying number of jobs

In the first series of tests, we vary the number of jobs. In Figure 3.11a, we can see that the Distributed Online runs very close to Two Phase and that they both achieve near-optimal results, though the approximation ratios drop from about 1 to 0.9 as we increase the number of jobs. In contrast, $m$-Admission and the Centralized Online algorithm performed less well (with ratio about 0.8). The reason lies in the greedy choice Admission makes, first considering scheduling an earliest-finishing job, rather than basing the choice on job weights. The result could be that many jobs are scheduled on machines where their weights are low. The Two Phase algorithm is not affected by the varying weights because when it pushes job instance to the stack, it compares the weights of overlapping jobs. Distributed Online performs well because it picks the most valuable job competing for the current timeslots. We also notice that Centralized Online performed

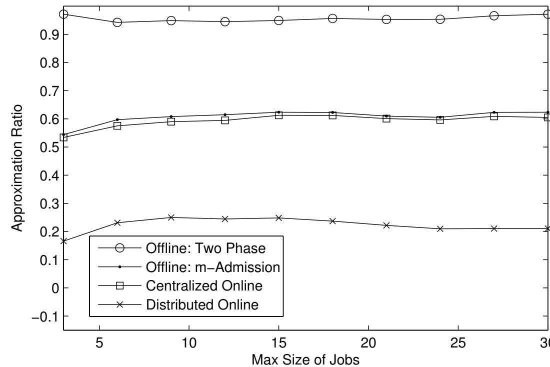
slightly better than $m$-Admission. The reason is that Centralized Online uses Global Admission instead of $m$-Admission and schedules jobs on all machines in parallel. $m$-Admission schedules jobs machine-by-machine, and so many jobs with high initial weights are scheduled on the first several machines we consider whereas most jobs are scheduled later, with low weights. Global Admission avoids this problem since jobs with high initial weight can be scheduled on all machines at a very early stage.

The results of the second test (see Figure 3.11b) are very similar to those shown in Figure 3.11a. This indicates that the algorithms performed similarly as long as jobs' weights decrease.




(a) Decreases linearly.

(b) Decreases following inverse cotangent function.




(c) Increases linearly.

Figure 3.12: Test on machine-dependent weights, varying the maximum job size

In the third test (see Figure 3.11c), Two Phase performed well; the two al-

gorithms based on Admission perform slightly worse but still do better than the performance implied by their approximation guarantee. They performed worse because although they do not consider job weights in the first place, they may preempt jobs based on their weights. But a job with large weight in the beginning is very likely to have a larger weight in the end; therefore, we should not preempt the current job and schedule the job with high weight later. The Distributed Online algorithm did much worse, with a solution quality of about 20% of the optimal. A possible explanation of this outcome is that when the weight increases, Distributed Online, which schedules the high-weight jobs as early as possible, works exactly the opposite way of what is expected, because those jobs with high weights are supposed to be scheduled as late as possible.

We also find that Distributed Online performed better when there were more jobs. Because when there are more jobs, the Distributed Online is supposed to perform worse in the uniform-weight case; this actually helps in the weight-increasing case as these two cases required the opposite ways of scheduling jobs. This may be verified by that Global Admission based Centralized Online performed worse than $m$-Admission in Figure 3.11c, because Global Admission also attempts to schedule jobs with high weights early.

To further evaluate these algorithms, we conducted another series of tests varying the maximum job size similar to those recounted in Section 3.5. As we can see from Figure 3.12, the performance varies as we expect: the approximation ratio for Two Phase and Distributed Online falls and then rises again; the Admission-based algorithm performed less well with decreasing weights and even worse with increasing weights; Distributed Online performed the worst with increasing weights; Global Admission performed better than $m$-Admission with decreasing weights while worse than $m$-Admission with increasing weights.

In summary, Two Phase produces near-optimal results in different situations, regardless of whether the weights increase or decrease; the two Admission-based algorithms are not suitable for the machine-dependent weight case, even though they still guarantee a 2-approximation; finally, our Distributed Online algorithm performs well on decreasing weights but poorly on increasing weights.

Table 3.6 summarizes the performance of these algorithms under various circumstances. We can see Two Phase is good for general use because it pro-

duces good or near-optimal results in all these situations. $m$-Admission performed equally well except for the time-varying weights case. Because $m$-Admission run iteratively, Centralized Online performed similarly to $m$-Admssion; the online nature makes it hard to reach the offline algorithms' performance, however. Distributed Online performed well in all the cases except for the increasing weights setting, and it is the only one that naturally fits for the varying-bandwidth case.

| | Two Phase | $m$-Admission | Centralized Online | Distributed Online |
|---|---|---|---|---|
| Typical Instances | Near-opt | Near-opt | Near-opt | Good |
| Special traffic | Near-opt | Near-opt | Good | Good |
| Varying bandwidth | Good (BW estimation) | Good (BW estimation) | Good (BW estimation) | Good |
| Varying weights – decreasing | Good | Average | Average | Good |
| Varying weights – increasing | Good | Average | Average | Poor |

Table 3.6: Performance Comparison

## 3.7 Conclusion

We studied one class of scheduling problems in which jobs have machine-dependent release times and deadlines. This problem is motivated by scenarios in which data is delivered to mobile clients as they travel. We introduced different system architectures for different problem settings, and we adapted and proposed algorithms to solve this problem near-optimally or approximately. The performance evaluation showed that the algorithms performed well, even when applied to more realistic cases in which network bandwidth changes over time. In the end we explored the case in which job weights changes over time and we use the algorithms to solve it by interpreting time-varying weights as machine-dependent weights.

Several interesting open problems are raised by our work:

1. A BS may have information about its neighbors, or several close BSs could work jointly in a group to improve the Distributed Online algorithm.

2. The Centralized Online algorithm works iteratively, each time rescheduling most jobs including those having reservations. Overhead could be reduced by an incremental algorithm that attempts to limit the number of revisions made to the existing schedule.

# Chapter 4

# Replica Placement Problem in Cloud-based CDNs

Content Distribution Networks present another information-centric networking scenario, where the replica is provisioned once on a replica site and accessed by users multiple times. In order to enforce a Quality of Service requirement, users should be sufficiently close to a replica site. The challenge is how to pick and provision replica sites under the pay-as-you-go cost structure of the cloud.

## 4.1 Introduction

Traditional Content Distribution Networks (CDNs) such as Akamai and Mirror Image have deployed tens of thousands of data centers and edge servers to deliver content across the globe. It has become financially prohibitive for smaller providers to compete on a large scale following the traditional model by building new data centers.

The recent emergence of storage cloud providers such as Amazon S3, Nirvanix and Rackspace has opened up new opportunities to provision cost-effective CDNs. Storage cloud providers operate data centers that can offer Internet-based content storage and delivery capabilities with the assurance of service uptime and end user perceived service quality. Service quality is typically in the form of bandwidth and response time guarantees [53].

Two opportunities arise from the emergence of storage cloud services. First,

one can build a CDN serving others without the high cost of owning or operating geographically dispersed data centers. MetaCDN [54] is an example falling into this category. Second, small Web sites can build their own global CDN by simply becoming customers of multiple storage cloud providers operating in different continents and locales. We refer to this new breed of CDN that is based on storage clouds as a "cloud CDN", as opposed to a "traditional CDN".

Storage cloud providers charge their customers by their storage and bandwidth usage following the utility computing model [55]. Storage cost is measured per GB per unit time and bandwidth cost is measured per GB tranferred.Bandwidth cost further consists of two components: upload cost for incoming data (e.g., provisioning of Web content) and download cost for outgoing data (e.g., serving user requests).

As the customer, a cloud CDN may take advantage of the competitive prices offered by different cloud providers and reduce its own expense. Combined with the on-demand scaling feature of the cloud, cloud CDNs can easily adjust their storage and bandwidth usage based on demand and possibly trade decreased service quality for reduction in costs. A cloud CDN also multiplexes resources among multiple customers/Web sites like a traditional CDN. In other words, cloud CDNs can provide similar functionality as traditional CDNs, but without actually owning any infrastructure.

However, as the only cost for cloud CDNs is the bandwidth and storage cost, they are very sensitive to the usage variations. To efficiently operate a cloud CDN, intelligent replica placement and user redirection strategies are required. The "Replica Placement" problem in the traditional CDN setting has been widely studied in the literature. However, existing results cannot be readily applied to the cloud CDN setting for the following reasons:

1. Many works in traditional CDNs assume that the network topology is given, such as a tree rooted at the origin server. However, in cloud environment, we have the freedom to build any topology among all the potential replica sites. This topology may be different from the underlying network topology. Thus, replica placement in cloud CDN is a joint problem of building distribution paths and replication.

2. The cost of provisioning a replica site $v$ from another site $u$ has been considered as a distance $d(u, v)$ between $u$ and $v$. For traditional CDN, usually the edges are undirected, meaning $d(u, v) = d(v, u)$. However, storage clouds charge different prices for uploading and downloading, which requires the edge to be directed. This implies that only choosing a set of replica sites is not good enough; we need to specify the replication directions.

The content of the Web site is maintained on an origin server. The cloud CDN rents a set of storage cloud sites where replicas are placed. As today's Web sites consist of segments (text, image or video) with different popularities and geographic affinities, which may require different SLAs, we consider each segment individually and solve the problem of replicating a segment of a web site to cloud sites.

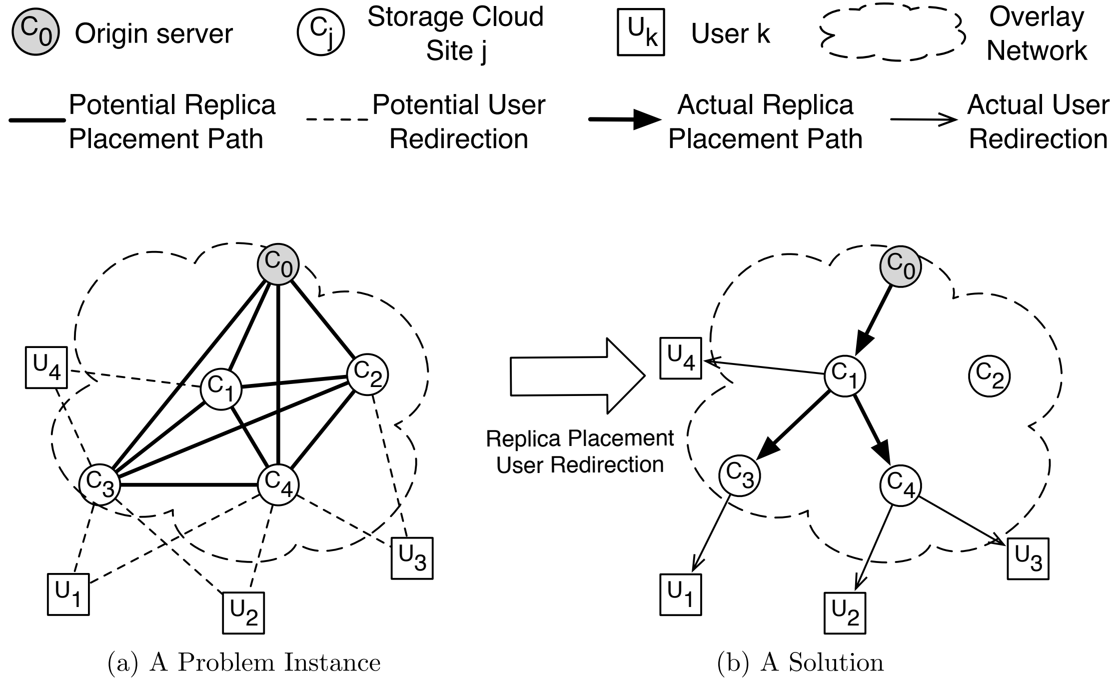
Request redirection is an integral part of replica placement. Any replica placement algorithm requires the specification of the corresponding request redirection strategy. Once replicas are in place, user requests can be redirected from the origin server to the appropriate replicas using any of the approaches in a traditional CDN setting, such as URL-rewriting, DNS-based request redirection, or transparent interception of user requests [56].

We show an example scenario of replica placement in a cloud-based CDN in Figure 4.1. In an overlay network shown in Figure 4.1a, potential replica distribution paths are drawn as bold lines among an origin server $C_0$ and the potential replica sites $C_1$–$C_4$. We assume each site has a routing path to each user $U_k$[1]. However, only a subset of these paths satisfy QoS requirements for user requests, which are drawn as dashed lines. An example solution of the replica placement problem is shown in Figure 4.1b, where $C_3$ is chosen to serve requests from $U_1$ and $C_4$ is chosen to serve requests from $U_2$ and $U_3$. $C_1$ is chosen to server requests from $U_4$ as well as relay the replica from $C_0$ to $C_3$ and $C_4$.

The total cost for this solution includes storage, upload and download cost at $C_0$, $C_1$, $C_3$ and $C_4$. For a replica site that serves user requests (e.g., $C_3$ or $C_4$), its upload cost is incurred by incoming traffic to provision Web content at the node, and its download cost is incurred by outgoing traffic to serve user requests[2].

[1] It is well known that the AS routing path is not necessarily the shortest path.

[2] We ignore bandwidth cost associated with Web requests since its volume is negligible com-




(a) A Problem Instance (b) A Solution

Figure 4.1: Problem Scenario

We treat the origin server $C_0$ as a regular node in the storage cloud. Its upload cost is incurred by provisioning of Web content, and its download cost is incurred by traffic from $C_0$ to $C_1$. For a replica site that relays replicas to other sites, for example $C_1$, its download cost is also incurred by traffic to provision other replicas. Storage cost is incurred by storing the entire replica, including the cost at origin server $C_0$ and all provisioned replica sites, $C_1$, $C_3$ and $C_4$.

In this chapter, we propose a suite of algorithms for replica placement for a cloud CDN. A cloud CDN is built particularly for a Web site (or a Web site segment, in the case that each segment has a different SLA), whose owner signs a SLA with the cloud CDN. In the SLA, certain QoS requirements are specified, such as $X\%$ of end user requests will incur response time of less than a certain value $Y$ from a certain region. The goal of the cloud CDN is to minimize its cost for hosting the Web site while honoring the SLA. Via trace-based study, we show that cloud CDN significantly reduces the monthly cost of providing CDN service to small Web sites to as low as 2.62 US Dollar compared to the 99 US Dollar

pared to that of the corresponding replies. As video traffic begins to dominate Web traffic, request traffic volume is becoming even less compared to reply traffic volume.

minimum charge by a traditional CDN.

We first present an Integer Programming (IP) formulation of this problem and prove that it is NP-hard. We then propose two sets of heuristics to place replicas: (1) offline and static algorithms based on past user request patterns and (2) online-static and online-dynamic algorithms triggered by each request. We evaluate the performance of our heuristics via Web trace based-simulation. To the best of our knowledge, this is the first study into the replica placement and distribution path construction problem in cloud CDN settings.

The rest of this chapter is organized as follows. Section 4.2 presents related work. Section 4.3 formally defines the problem. Section 4.4 presents two heuristic algorithms for the offline settings while Section 4.5 proposes three heuristics for the online setting. All algorithms are then evaluated in Section 4.6. Finally, Section 4.7 concludes the chapter.

## 4.2 Related Work

A considerable amount of research has been done for replica placement in CDNs. The cost model has evolved to include one or more of the three types of costs: retrieval (or download), storage and update (or upload) cost.

In terms of minimizing content retrieval cost only, Li et al. [57] and Krishnan et al. [58] showed that replica placement in general network topologies is NP-complete and provided optimal solutions for tree topologies. Qiu et al. [5] evaluated a number of heuristics and found a greedy algorithm offering the best performance. Radoslav et al. [59] and Jamin et al. [60] proposed a fan-out based heuristic in which replicas are placed at nodes with the highest fan-out. In these proposals, however, once a set of replica sites is chosen, the distribution paths are more or less implied.

In addition to retrieval cost, Xu et al. [61] and Jia et al. [62] further added update cost, whereas Cidon et al. [63] added storage cost into consideration. Furthermore, Kalpakis et al. [64] comprehensively considered all three costs (retrieval, update and storage) and offered solutions for a tree topology only. However, none of the work studied the case in which provisioning cost between replica sites is relevant to the replication direction.

Enhancing results from the above studies, a rich body of work has added QoS

into consideration by requiring all user requests to reach replica servers within a certain network distance. Tang et al. [7] and Wang et al. [8] proposed algorithms to optimize total storage and update cost. They used the assumption that requests can be issued from any node, and ignored retrieval cost. Rodolakis et al. [6] added server capacity limitation to the formulation while optimizing storage and retrieval cost. Contrary to existing solutions, our schemes also consider the online case in which we allow occasional violations of QoS allowed by the SLA.

All the solutions described above are static in nature in that they either assume requests originate uniformly from all nodes or they simply use past request patterns to make replica placement decisions. Bartolini et al. [65] took a different approach by modeling replica placement as a Markovian decision process, and proposed a centralized heuristic. Presti et al. [66] further developed a distributed heuristic. Vicari et al. [67] optimized replica placement and traffic redirection in order to balance request load on replicas. Loukopoulos et al. [68] solved the problem of transferring one set of replica placement into another. In addition to CDN, the replica placement problem has also been studied in Overlay [8], Grid [69] and P2P settings [70].

MetaCDN by Broberg et al. [54] is a low cost CDN using storage cloud resources. The system provides mechanisms to place content in different storage cloud provider networks and redirect user requests to appropriate replicas. However no replica placement and request redirection algorithm is given. Our algorithms can be readily used by MetaCDN to make the system comprehensive.

## 4.3 Problem Formulation

In this section, we first define different problem settings and then formulate the offline problem as an Integer Program. In the end, we define two basic operations used in our heuristics.

### 4.3.1 Problem settings

We study the replica placement problem in both offline and online settings. In the offline setting, all input parameters are known, including user request patterns.

Algorithms for this setting do not react to individual user requests, but rather aggregate all requests from one user into one request.

In the online setting, we assume all input parameters to replica placement are known except future requests. This implies that we can only consider requests in real time and our decision should be made based on each request.

An algorithm can be either static or dynamic. We define a static algorithm as one that computes the set of replica sites based on current knowledge: past requests for online settings and future requests for offline settings. Then regardless of user requests, we maintain the set of provisioned replica sites. We define a dynamic algorithm as one that may open new sites for future requests if our previous decision was inefficient.

The offline algorithms are static in nature, while the online algorithms are either static, using past user request patterns to derive replica placement or dynamic, making placement decisions reacting to real time user requests. The characteristics of different problem settings are summarized in Table 4.1.

<table>
<tr><th colspan="2">Problem Setting</th><th>Characteristics</th></tr>
<tr><td colspan="2">Offline</td><td>Aggregate requests, no new sites</td></tr>
<tr><td rowspan="2">Online</td><td>Static</td><td>Individual request in real time, no new sites</td></tr>
<tr><td>Dynamic</td><td>Individual request in real time, allow new sites</td></tr>
</table>

Table 4.1: Problem Settings

### 4.3.2 IP formulation

Now we formulate the offline problem of replicating a Web site replica/segment hosted by an origin server $C_0$ at multiple storage sites to serve all end user requests. In reality, a Cloud provider owns and operates multiple storage sites or data centers; however, each site belongs to only one provider. Cloud sites from different providers may be co-located, but they may offer different prices as well. We assume there are $m$ end users $\mathcal{U} = \{U_1, U_2, ..., U_m\}$ indexed by $k$, and $n$ storage cloud sites $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ indexed by $i$ or $j$.

We assume the size of the replica is $W$ bytes. As mentioned in Section 4.1, a storage cloud site will charge for storage, incoming and outgoing traffic. For storing

the replica, the site $C_j$ will charge unit storage cost of $S_j$ per GB[3]; for uploading the replica onto $C_j$, $C_j$ will charge a unit price of $P_j$ per GB for incoming traffic; for distributing the replica to other sites or delivering content to end users, $C_j$ will charge unit price of $D_j$ per GB for outgoing traffic.

Let $V_{uv}$ denote the replication cost from $u$ to $v$. Depending on the nature of the node $v$, $V_{uv}$ has two different meanings. In the first case, $v$ is a cloud site, $V_{uv}$ is the site opening cost also known as the cost of provisioning a site $v = C_j$ by downloading the replica from any site $u = C_i$ that has the replica. The origin sever will update the replica from time to time. On average, we assume that a fraction $F \cdot W$ of the content needs to be updated per unit time, where $F$ indicates the update frequency. Then $V_{uv} = V_{ij} = (S_j + P_jF + D_iF)W$. We generally assume $V_{ij} > 0$. However, the first site having the replica is the origin server. If we assume download price on the origin server is free, that is, $D_0 = 0$, every site would download the replica from the origin server; otherwise, in more general cases where download price per GB is not negligible at the origin server, the replica distribution paths will be a tree structure rooted at $C_0$.

In the second case, $v$ in $V_{uv}$ indicates a user, say $v = U_k$, $V_{uv}$ is the access cost of user $U_k$ who is assigned to site $u = C_j$. We assume that user requests are given in the offline problem. An end user $U_k$ has a request of size $w_k$ bytes and then $V_{uv} = V_{jk} = w_kD_j$. Note that $w_k$ may be less than $W$, because of fractional requests of the whole replica, (e.g. browsing only a portion of the replica/Web site segment) or use of a Web cache. It may be greater than or equal to $W$, because of repeated requests without caching.

We use $L_{jk}$ to denote the routing distance from site $C_j$ to end user $U_k$. Note that this distance captures communication quality between two nodes, and it can be in the form of either hop count or delay. In order to satisfy QoS requirements for end user requests, we need to limit this distance within a required QoS distance $Q$. We do not put bandwidth capacity as a hard constraint on replica sites. Instead, we restrict $L_{jk}$ to be bounded by a smaller value of QoS distance $Q$. This is based on the assumption that web traffic congestion has limited impact on storage clouds; one of the key benefits of cloud storage providers like Amazon is their ability to

[3]In practice, storage cost is charged per unit time where the unit is in the order of months; storage cost is in the form of a step function of length of time considered. For the time duration considered in each problem instance, we pre-compute the storage cost as a per GB value.

add capacity on demand quickly when replica sites become congested[4]. When such congestion happens, $L_{jk}$ increases but is still bounded by $Q$.

The objective of the replica placement problem is to find a replication strategy minimizing the total cost by provisioning a subset of the cloud sites such that each end user is assigned to at least one site and the QoS requirements for end user requests are satisfied. Formally, let $G = (N,\ E)$ be a directed graph, where $N = \{C_0\} \bigcup \mathcal{C} \bigcup \mathcal{U}$. Edge $(C_i, C_j) \in E$ indicates a feasible provisioning or replication path from $C_i$ to $C_j$. Edge $(C_j, U_k) \in E$ indicates a potential assignment of user $U_k$ to $C_j$; in other words, for any $(C_j, U_k) \in E$, $L_{jk} \leq Q$. Each edge $(u, v) \in E$ is associated with a cost $V_{uv}$. The goal is to find a subtree that contains a directed path between $C_0$ and every $U_k$. The problem is formally defined as:

$$\min \sum_{(u,v)\in E} y_{uv} V_{uv} \tag{4.1}$$

subject to

$$\sum_{w\in N} x^k_{uw} - \sum_{v\in N} x^k_{vu} = \begin{cases} 1, & u = C_0 \\ -1, & u = U_k \\ 0, & \text{otherwise} \end{cases} \qquad \forall\ U_k \tag{4.2a}$$

$$x^k_{uv} \leq y_{uv} \qquad \forall\ (u, v) \in E \tag{4.2b}$$

where $y_{uv}$ is a binary variable indicating whether a replication path $(u, v)$ is chosen. And variable $x^k_{uv} \geq 0$ is the number of replication flows between $C_0$ and $U_k$ via $(u, v)$.

Constraints (4.2a) guarantees that there is a directed path from $C_0$ to every $U_k$. Constraints (4.2b) indicates that a flow is allowed only if its replication path is chosen in the solution.

**Theorem 1.** *Problem (4.1) is NP-hard and cannot be approximated with factor better than $O\log(n)$, where $n$ is the number of users.*

[4]As traffic load approaches network and server capacity in the cloud provider network, delay is bound to increase. In future work, we plan to add link capacity and site capacity constraints into the formulation.

*Proof.* Given is an instance of the minimum set cover problem, which cannot be approximated in polynomial time to within a factor of $\frac{1}{2}\log_2(n)$ [71]. Assume there are $n$ elements in the whole set $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ and $m$ subsets of $\mathcal{C}$ in $\mathcal{P} = \{P_1, P_2, ..., P_m\}$. Construct a directed graph $G = (V, E)$ with $n+m+1$ nodes: $V = \{r, C_1, C_2, ..., C_n, P_1, P_2, ..., P_m\}$ and $E = \{(r, P_j)|\ \forall j\} \cup \{(P_j, C_i)|\ \forall i, j, C_i \in P_j\}$. An instance of Problem (4.1) is then constructed on graph $G$ by assuming that $r$ is the origin server, $C_i$'s are users and $P_j$'s are potential replica sites and setting $O_j = 1$, $w_i = 1$, $D_j = 1$ and $Q = 1$. QoS constraints are satisfied within one hop so that $C_i$ can be assigned to $P_j$ only if $(P_j, C_i) \in E$.If there is a solution to the instance of Problem (4.1), every user $C_i$ is connected to a $P_j$ within one hop with minimum opening cost and connection cost, which means every element in $\mathcal{C}$ is covered and with the minimum number of subsets $P_j$. This completes the proof. □

We implement the IP formulation (4.1) and solve it with CPLEX [47]. The typical running time for the solver ranges from minutes to hours which is too long to be practical. This also motivates the need for heuristics. However, for a reasonably sized problem instance, we still solve it optimally to obtain a lower bound to evaluate our algorithms.

The notations used in this chapter are summarized in Table 4.2.

| **Term** | **Definition** | **Term** | **Definition** |
|---|---|---|---|
| $U_k$ | User $k$ | $w_k$ | Request size of $U_k$ |
| $C_j$ | Cloud site $j$ | $W$ | Replica size |
| $C_0$ | Origin server | $V_{ij}$ | Replication cost $i \to j$ |
| $S_j$ | Storage cost of $C_j$ | $L_{jk}$ | Distance from $C_j$ to $U_k$ |
| $D_j$ | Download cost of $C_j$ | $Q$ | QoS distance |
| $P_j$ | Upload cost of $C_j$ | $F$ | Update Frequency |

Table 4.2: Summary of Notations

### 4.3.3 Basic operations

Before we look into the heuristics for various settings, we need to define two basic operations: *Replica Provision* and *User Redirection*.

The *Replica Provision* operation for a cloud site $C_j$ includes downloading a replica from a proper site $C_i$ and uploading and storing the replica on $C_j$. We also refer to this operation as *open site*. The first step is to locate the source site $C_i$. A greedy strategy is to download from one of the currently opened sites that has the lowest download price per GB. Let $O_j$ denote the opening cost of $C_j$. Then

$$O_j = \begin{cases} 0 & \text{if } C_j \text{ is OPEN} \\ \min_{i \in \{i | C_i \text{ is OPEN}\}} V_{ij} & \text{otherwise} \end{cases}$$

Note that the order in which replica sites are opened affects the cost for downloading replica to a site, because the download source can only be selected from the currently opened sites.

The second operation is *User Redirection* or *User Assignment*. In the offline setting, all requests of a user are directed to the best replica site. In the online setting, each user request is redirected separately. In practice, if a site is overloaded, attacked or fails, we remove the site and either rerun the offline algorithms or continue with iterations of the online algorithms. As a result, requests from the same user are redirected to multiple sites over time. Note that it is common practice in traditional CDNs today to redirect user requests to different sites upon congestion or failure of the current site the user is directed to.

## 4.4 Offline Algorithms

In this section, we present two heuristic algorithms *Greedy Site* and *Greedy User* for the offline setting.

### 4.4.1 Greedy Site (GS)

The Greedy Site (GS) algorithm is adopted from an approximation algorithm for the Set Covering Problem [72]. We name it Greedy Site (Algorithm 9) because we iteratively decide to open a closed site which has the maximum utility and assign all its potential users to it. The site utility is defined as the total request volume (in bytes requested) divided by the total cost incurred by serving these requests. The potential users of a site are the users who are within the QoS distance of this

site but not yet assigned. We open this site, then find the next best site to open until all users are assigned to a site.

**Algorithm 9** Greedy Site (GS)

1: $E$ is the set of users who have not been assigned.
2: $E_j$ is the current set of users who can be assigned to $C_j$.
3: **while** $E \neq \emptyset$ **do**
4: $\quad W_j \leftarrow \sum_k w_k, U_k \in E_j$.
5: $\quad j^* \leftarrow \underset{j \in \{j | C_j \text{ is CLOSED}\}}{\text{argmax}} \frac{W_j}{W_j D_j + O_j}$
6: $\quad$ Assign all users in $E_{j^*}$ to $C_{j^*}$
7: $\quad$ Open $C_{j^*}$
8: $\quad E \leftarrow E - E_{j^*}$,
9: **end while**

### 4.4.2 Greedy User (GU)

We name the second algorithm Greedy User (GU, Algorithm 10) because users rather than sites are evaluated one after another. In the algorithm, we assign users to sites with the lowest cost and open new sites if necessary. We consider users with the least number of potential sites first (line 2 in Algorithm 10). This is because the more sites we can assign a user to, the more likely we can assign the user to a site already opened. Therefore, we avoid opening unnecessary sites. Note that $O_j$ is non-zero only when $C_j$ is CLOSED, as defined in Section 4.3.3.

**Algorithm 10** Greedy User (GU)

1: $F_k$ is the set of sites that $U_k$ can be assigned to.
2: Sort $\mathcal{U}$ in increasing order of $|F_k|$
3: **for all** $U_k$ **do**
4: $\quad j^* \leftarrow \underset{j \in \{j | C_j \in F_k\}}{\text{argmin}} (O_j + w_k D_j)$
5: $\quad$ Assign $U_k$ to $C_{j^*}$
6: $\quad$ **if** $C_{j^*}$ is CLOSED **then**
7: $\quad\quad$ Open $C_{j^*}$
8: $\quad$ **end if**
9: **end for**

## 4.5 Online Algorithms

In this section, we present three online heuristics. We first present the basic algorithm Greedy Request (GR, Algorithm 11) where each request is processed upon arrival. For the first request of a user, we select the lowest cost potential site for the user based on the total cost for serving this request, opening the site if necessary. Any further requests from the same user will be redirected to the assigned site for the user. It is similar to Greedy User, except the order to evaluate users follows the arrival order for the first request of each user, and only the volume for the first request is considered for each user in making replica provision and user redirection decisions. Note that Greedy Request (GR) works in both offline and online settings.

**Algorithm 11** Greedy Request (GR)

1: **for all** request $q$ in increasing arrival time **do**
2:    Request $q$ is from $U_k$ and of size $w_q$
3:    **if** $U_k$ is assigned to $C_j$ **then**
4:       Redirect $q$ to $C_j$
5:    **else**
6:       $F_k$ is the set of sites that $U_k$ can be assigned to.
7:       $j^* \leftarrow \underset{j \in \{j | C_j \in F_k\}}{\operatorname{argmin}} (O_j + w_q D_j)$
8:       Redirect $q$ to $C_{j^*}$
9:       **if** $C_{j^*}$ is CLOSED **then**
10:          Open $C_{j^*}$
11:       **end if**
12:    **end if**
13: **end for**

In the online setting, one can use results from past request patterns to make replica placement decisions using offline algorithms, and only redirect user requests to the already opened sites. As shown in Section 4.6, Greedy Site (GS) performs better than Greedy User (GU), therefore we select Greedy Site (GS) to use in our online algorithms.

Based on different combinations of Greedy Request (GR) and Greedy Site (GS), we have the following three online schemes: *Greedy Request Only (GRO)*, *Greedy Request with Preallocation (GRP)* and *Greedy Site Only (GSO)*.

### 4.5.1 Greedy Request Only (GRO, dynamic)

One way to handle online requests is to use Greedy Request (GR) only, and open sites when needed in a dynamic fashion. Note that site opening operation is always triggered by the first request of a user. In practice, this request is sent to the origin server $C_0$ and served by $C_0$, while the replica is being provisioned at the chosen site. Only after a site has been opened and provisioned with replica, requests can be redirected to the site. During the interval between the time a replica site is chosen and the time it is provisioned with replica, all requests will be served by $C_0$. We assume any request served by $C_0$ violates QoS constraints in our problem formulation. We use the term *QoS violation* to refer to the phenomena of requests being redirected to a site violating QoS constraints.

### 4.5.2 Greedy Request with Preallocation (GRP, dynamic)

In order to reduce QoS violations caused by dynamically opening all required sites, we preallocate some sites based on past request patterns. We use Greedy Site (GS) based on recent user requests to derive a set of sites, and provision replicas on them. Upon the arrival of a new request, we execute Greedy Request (GR) dynamically. If a site chosen for the first request for a user has already been opened, then the request can be served from the site without any QoS violation. Only when such a site is closed, will it result in a QoS violation. When there is little variation in user request patterns and one can predict future requests based on history, GRP performs well without many QoS violations as shown in Section 4.6.

### 4.5.3 Greedy Site Only (GSO, static)

The two schemes above are dynamic, since regardless of preallocation, when a request arrives and can not be served by existing open sites, we open new sites. However, we can fix the set of open sites determined by Greedy Site (GS) based on request history, and redirect all future requests to existing open sites only. The Replica Provision process is therefore static. User Redirection in this case is slightly different: when there are preallocated open sites within the QoS distance, we assign the user to the site with the least cost; otherwise, we assign the user to

the closest site regardless of cost. Note that assigning users to sites with distance larger than QoS distance results in QoS violations.

## 4.6 Performance Evaluation

In this section, we evaluate our heuristics via Web trace-based simulation.

### 4.6.1 Potential replica sites, request pattern and cost

The information on potential replica sites is extracted from the iPlane Internet PoP topology [73], which provides PoP-node-to-IP mappings and inferred links between PoP nodes with latency information. For each problem instance we evaluate, we randomly pick one PoP node as the origin server $C_0$ and a subset of PoP nodes as cloud sites.

We then place the group of chosen PoP nodes including the origin server and the cloud sites onto a metric field using the geographical information of each node. To place a node, we first look up its IP address in the GeoLite city database [74] and convert the IP address to its corresponding (latitude, longitude) pair. However, a node may be associated with multiple IP addresses. In this case, we estimate its location with a technique similar to that proposed in [75].

We vary the number of cloud sites from 20 to 48. We choose such a small number because it reflects the fact that there are few cloud providers today. However, simulations with 200-300 cloud sites (results not shown here) confirm that the performance with a large number of sites is similar to that with a small number.

We extract end user request patterns from Penn State CSE Web access traces[5], which includes the host name and IP address information between Oct 1st and Dec 31st in 2010. In the 92-day trace, there are 1,186,213 user requests with an average request size of 25 KB and 268 MB requests per day. We sum all the bytes for distinctive Web page URLs to derive the default Web site replica size of 175 MB. By default, we run simulations for 30 days using traces from Nov. 1–30, 2010; we always use Nov. 1st as day one when we vary the trace period in simulations. The 30-day trace from Oct. 2–31st is used for preallocation in online settings.

[5] http://www.cse.psu.edu/

Using end user IPs from web server access traces, we map end users onto the same metric field with the origin server $C_0$ and the cloud sites. Then we assign distance value $L_{jk}$ from cloud sites to users. Padmanabhan et al. [76] contend that in recent years there is significant correlation between network delay and geographic distance. Because we do not have actual hop count information between users and cloud sites, we use geographic distance as an indicator of delay. In fact, the choice of distance metric does not impact the performance of our algorithms; any distance metric that is capable of describing the QoS requirement is applicable. In our simulation, we set the default QoS distance to be 5 (about 350 miles converted to geographical distance). A value too small does not guarantee that every user has at least one feasible site while a value too large makes every site fall within the QoS range which essentially renders the QoS constraint ineffective.

To summarize, in our simulations, the origin server, cloud sites and end users are located in the same metric field. The distance $L_{jk}$ from $C_j$ to $U_k$ is the geographic distance.

Typical storage cloud costs in 2010 are shown in Table 4.3. We can see that for today's cloud provider sites, unit price for storage, upload and download is comparable. As we need to evaluate more than four cloud sites, we use these typical costs as a guideline and randomly group them to generate new price combinations for different cloud sites. For example, we may use incoming cost of Nirvanix USA/EU, outgoing cost of Amazon S3 USA and storage cost of Amazon S3 EU as the price for one cloud site in our simulation.

| Cost Type | Nirvanix USA/EU | Nirvanix Global | Amazon S3 USA | Amazon S3 EU |
|---|---|---|---|---|
| Incoming data ($/GB) | 0.18 | 0.18 | 0.10 | 0.10 |
| Outgoing data ($/GB) | 0.18 | 0.18 | 0.17 | 0.17 |
| Storage ($/GB/month) | 0.18 | 0.25 | 0.15 | 0.18 |

Table 4.3: Typical Storage Cloud Prices in 2010

We evaluate the performance of the heuristics by varying the following four parameters:

1. replica size

2. length of trace period

3. QoS distance

4. number of sites

where each one has a range of ten values. Table 4.4 summarizes the default parameter values and the parameter ranges.

| | Replica size | $Q$ | Trace period | # of sites |
|---|---|---|---|---|
| Default | 175MByte | 5 | 30 days | 20 |
| Range | 10MB–10GB | 5–14° | 20–56 days | 20–48 |

Table 4.4: Simulation Parameter Summary

We vary one parameter at a time while fixing the rest at their default values. This results in 40 parameter settings. For each parameter setting, we generate 50 problem instances. This results in $50 \times 40 = 2000$ total problem instances. For each instance, we first randomly select from the iPlane topology, a subset of the nodes to be cloud sites and the origin server; we then assign a random combination of prices to each site. The length of trace period determines the storage cost, which is the monthly cost for any period up to 31 days and twice the monthly cost for any period from 32 to 61 days.

For each problem instance, we obtain the optimal cost by solving it using the CPLEX solver [47]. Figure 4.2 shows a CDF of the optimal costs in US Dollars in all tests. As we can see, 80 percent of the tests incurred costs less than 2.62

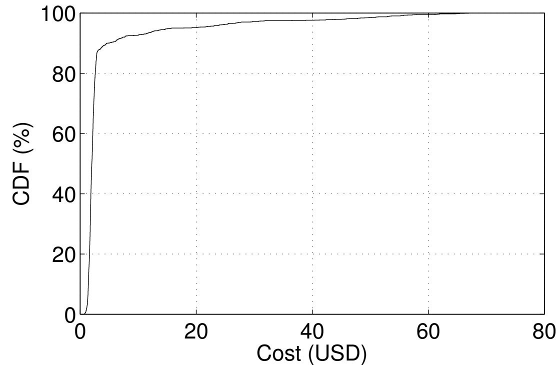



Figure 4.2: CDF of the Optimal Costs in All Tests

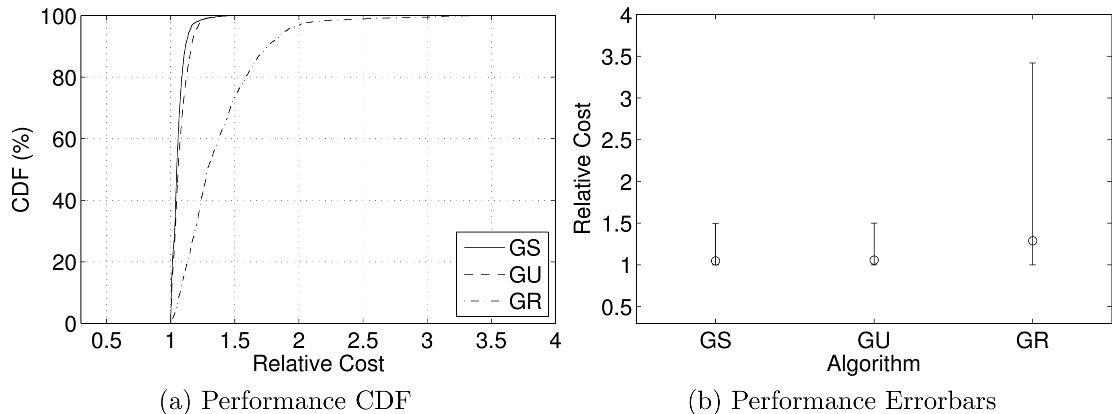
US Dollars, and almost all costs are below 60 Dollars. This is much less than the

pricing of a traditional CDN[6], whose minimum price per month is 99 Dollars[7]. Traditional CDNs may offer a lower per GB price for large Web sites with high volume of traffic. However, we argue that cloud CDN is more flexible and mainly focuses on benefiting small Web sites such as the one in our tests.

### 4.6.2 Performance of Offline Algorithms

We evaluate our offline heuristics by reporting *relative cost*, defined as ratio of total cost over the optimal cost.

Note that Greedy Request (GR) can be used as either an offline or an online algorithm, so we include it here for comparison. As shown in the CDF of relative cost (Figure 4.3a), Greedy Site (GS) and Greedy User (GU) perform well and achieve relative cost of 1.5 most of the time. GS is slightly better than GU which is better than GR. The 95th percentile value of the relative cost is 1.14, 1.20 and 1.91 respectively.




(a) Performance CDF (b) Performance Errorbars

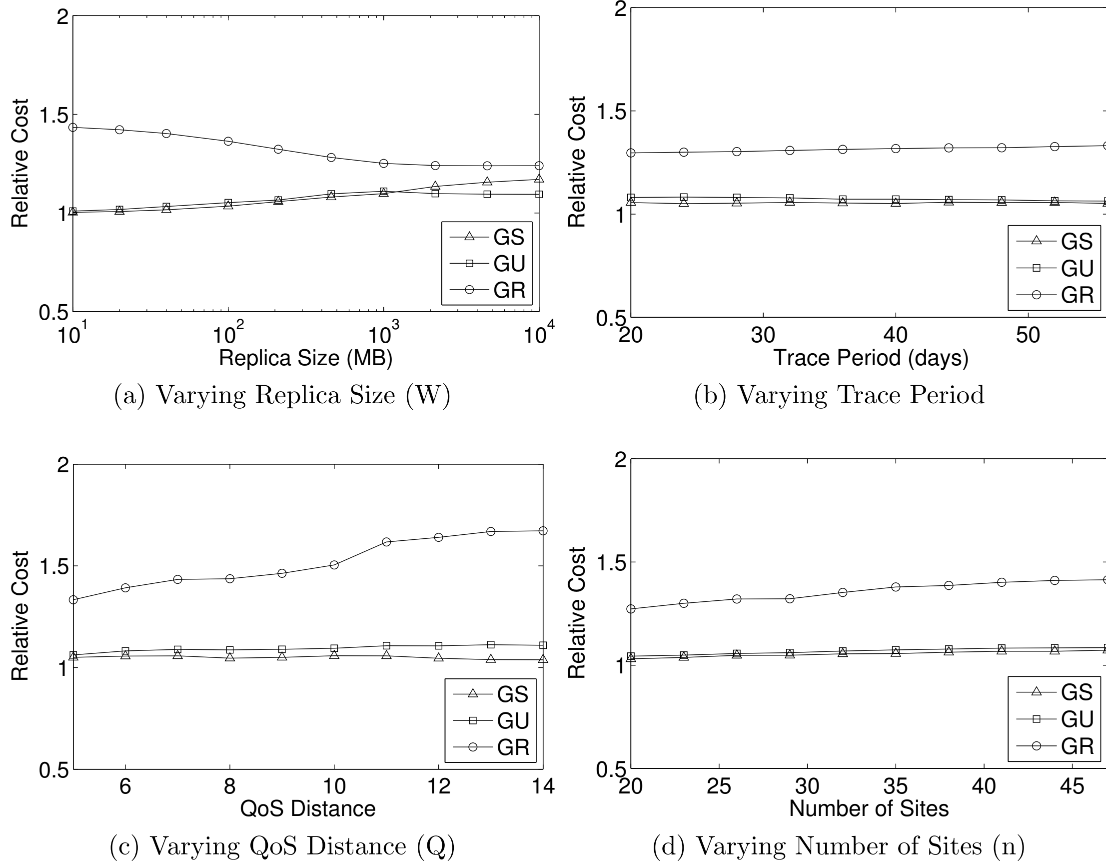
Figure 4.3: Performance Statistics for Offline Static Algorithms

We plot an error bar in Figure 4.3b using the max, min and median of relative cost for each algorithm. GR has the highest worst and median value for relative cost and GS is the most stable with the steepest CDF curve (Figure 4.3a) and the lowest median cost value of 1.05. We recommend GS as the best offline algorithm,

[6]CacheFly, http://www.cachefly.com/pricing.html

[7]Other major CDN providers such as Akamai do not publish their prices, but they generally require similar or even higher minimum monthly fee.

and use it to analyze past traces in order to conduct site preallocation for online algorithms.




(a) Varying Replica Size (W)

(b) Varying Trace Period

(c) Varying QoS Distance (Q)

(d) Varying Number of Sites (n)

Figure 4.4: Relative Cost for Offline Algorithms

Figure 4.4 demonstrates the effect of different parameters on the algorithm performance. To better understand the results, we look into the total cost structure.

Let $X^{OPT}W$ denote the optimal site opening cost and $\Delta XW$ denote the additional site opening cost incurred by an algorithm. Similarly, let $\sum_k Y_k^{OPT} w_k$ denote the optimal user access cost and $\sum_k \Delta Y_k w_k$ denote the additional user access cost. Then the relative cost plotted in the figures is:

$$\text{relative cost} = \frac{(X^{OPT} + \Delta X)W + \sum_k (Y_k^{OPT} + \Delta Y_k) w_k}{X^{OPT}W + \sum_k Y_k^{OPT} w_k}$$

In one extreme case, when $W >> \sum_k w_k$,

$$\text{relative cost} \approx \frac{X^{OPT} + \Delta X}{X^{OPT}} = 1 + \frac{\Delta X}{X^{OPT}},$$

only the additional cost for site opening is observable. In the other extreme case, when $W << \sum_k w_k$,

$$\text{relative cost} \approx \frac{\sum_k (Y_k^{OPT} + \Delta Y_k) w_k}{\sum_k Y_k^{OPT} w_k} = 1 + \frac{\sum_k \Delta Y_k w_k}{\sum_k Y_k^{OPT} w_k},$$

only the additional cost for user access is observable. In all other cases, the relative sizes of $W$ and $\sum_k w_k$ decides which part of non-optimality (for either site opening or user access) is dominating in the total relative cost.

When $W << \sum_k w_k$, the two offline algorithm achieve near-optimal performance (Figure 4.4a, relative cost $\approx 1$), which indicates that $\sum_k \Delta Y_k^{GS,GU} \approx 0$. But GR incurs even higher relative cost when $W << \sum_k w_k$ than it does when $W >> \sum_k w_k$ ((Figure 4.4a). This indicates that

$$\frac{\sum_k (\Delta Y_k^{GR}) w_k}{\sum_k Y_k^{OPT} w_k} > \frac{\Delta X^{GR}}{X^{OPT}}.$$

Then we observed that, when $W >> \sum_k w_k$, GS and GU have smaller relative cost than GR, which means

$$\frac{\Delta X^{GR}}{X^{OPT}} > \frac{\Delta X^{GS,GU}}{X^{OPT}}.$$

In other words, in Figure 4.4a we observed that GS and GU incur near-optimal user access cost and less site opening cost than GR; while GR produces non-optimal cost in both two parts and the user access part weighs more in its non-optimality.

We then increase the trace period from 20 to 56 days, which has the same effect as increasing the user request size. As shown in Figure 4.4b, relative costs for GS and GU decrease slightly while relative cost for GR increases slightly. We also vary update frequency and observe similar results to the one shown in Figure 4.4a, for increasing frequency has the same effect as increasing replica size.

In the third test shown in Figure 4.4c, we vary QoS distance from 5 to 14; in

the forth test in Figure 4.4d, we vary the number of sites from 20 to 48. These two tests illustrate an important factor that affects performance: the size of the solution space. When we increase the QoS distance, or increase the total number of sites, a user has more choices for site selection which increases the size of the solution space. A larger solution space makes the problem more difficult for our heuristics and therefore their relative cost increases. However, the performance of GS and GU vary only slightly within reasonable range of parameter values.

In summary, GU and GS both achieve near-optimal solutions in various problem instances. Instead of aggregating all requests from each user into one as in GU and GS, GR processes requests in the order they arrive and assigns users to a site solely based on its first request. This assignment order results in both more sites being opened and users being assigned to sites with higher download cost, which leads to higher relative cost than the other two heuristics.

### 4.6.3 Performance of Online Algorithms

For the two heuristics requiring site preallocation, namely, Greedy Request with Preallocation (GRP) and Greedy Site Only (GSO), we first allocate sites based on prediction of user request patterns. We use the immediate past history as an indication of future user behavior. Specifically, when we simulate with a trace period of $T$ days starting on Nov 1st, 2010, we use the trace from the past $T$ days ending on Oct 31st, 2010; when $T$ is larger than 30 days, we only use the 30-day trace ending on Oct 31st, 2010 for preallocation. Therefore, we set the storage cost to the monthly cost during the preallocation process.

For every test, we report QoS violation ratio, which is the ratio of request volume (in bytes) that experiences QoS violation to the total request volume. In practice, the QoS violation clause of a SLA can also be written in different forms. One example is to measure the percentage of user requests that violate QoS requirements. Another example is to measure the percentage of users that suffer from QoS violation. The metric we select, request volume violation, puts more weight on large requests.

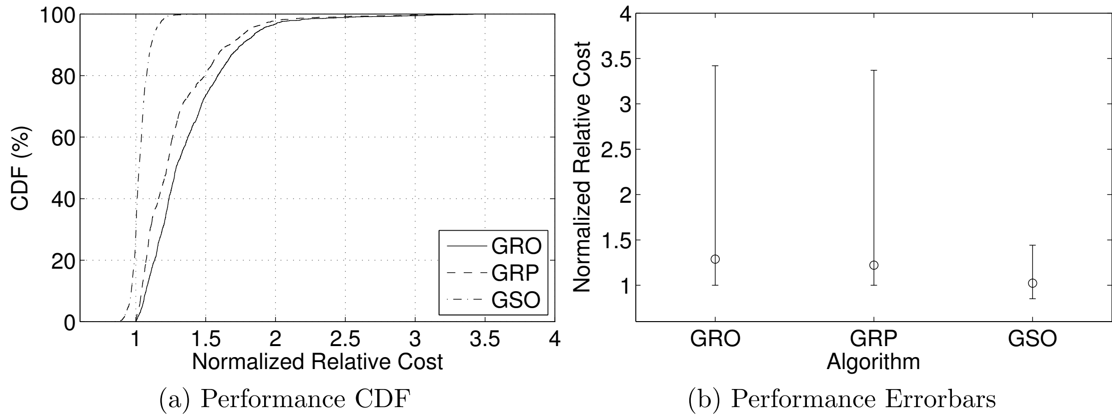
Rather than reporting relative cost, we introduce a new metric, *normalized*

*relative cost*, to evaluate online algorithms. It is defined as:

$$\text{normalized relative cost } = \frac{\text{relative cost}}{1 - \text{QoS violation ratio}}.$$

We introduce this metric in order to add penalty for each request resulting in a QoS violation. It combines relative cost and QoS violation ratio into one metric. However, for a reasonably small QoS violation ratio, relative cost and normalized relative cost result in very similar value.

The three online schemes in decreasing performance or increasing cost order are GSO, GRP and GRO. The 95th percentile value of the normalized relative cost is 1.15, 1.80 and 1.91 respectively, as shown in Figure 4.5a.




(a) Performance CDF

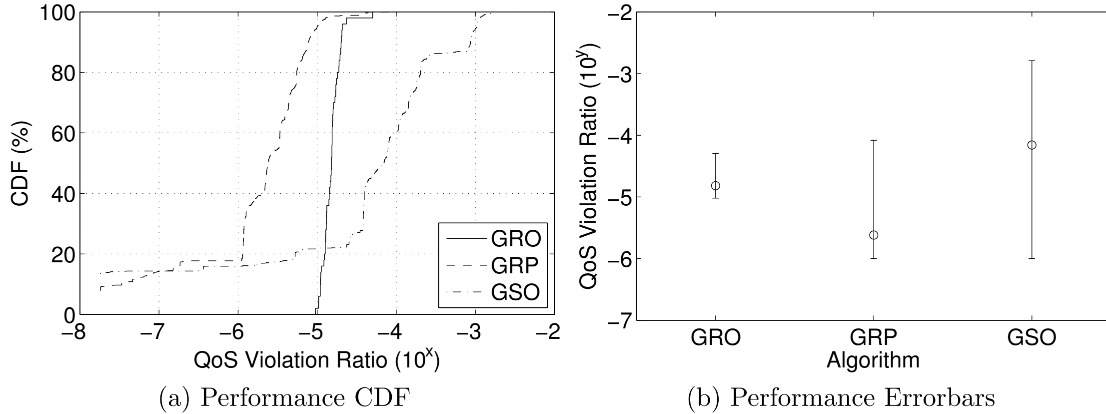
(b) Performance Errorbars

Figure 4.5: Performance Statistics for Online Algorithms

Out of the three heuristics, the static GSO has the lowest median cost very close to 1 (Figure 4.5b). It even results in lower cost than the offline optimal in 27% of the problem instances (Figure 4.5a). This is because GSO allows a user to be assigned to a site with a low downloading price while violating QoS constraints when no site in the preallocated set can meet the constraints. On the other hand, the offline optimal requires all QoS constraints to be satisfied, therefore it requires more sites to be opened than the solution by GSO.

Figure 4.6 shows statistics of QoS violation ratio of the three online heuristics. GSO has a much higher QoS violation ratio (median value around 0.7e-4), than the dynamic schemes (1.5e-5 for GRO and 2.4e-6 for GRP). GSO is the worst since no new sites can be opened, user requests are directed to preallocated sites

even with QoS violations. Clearly GSO trades lower cost with lower user QoS satisfaction. GRO is better as only the first request from a new user triggers QoS violations if a new site has to be opened for this user. GRP is the best in the sense that it eliminates the instances of QoS violations from GRO when the first request from a new user is redirected to a site already opened by the preallocation process. Furthermore, in GSO, users who experience QoS violations are permanently assigned to the sites that violate their QoS requirements. This causes unfairness among users and may be prohibited by some SLAs.



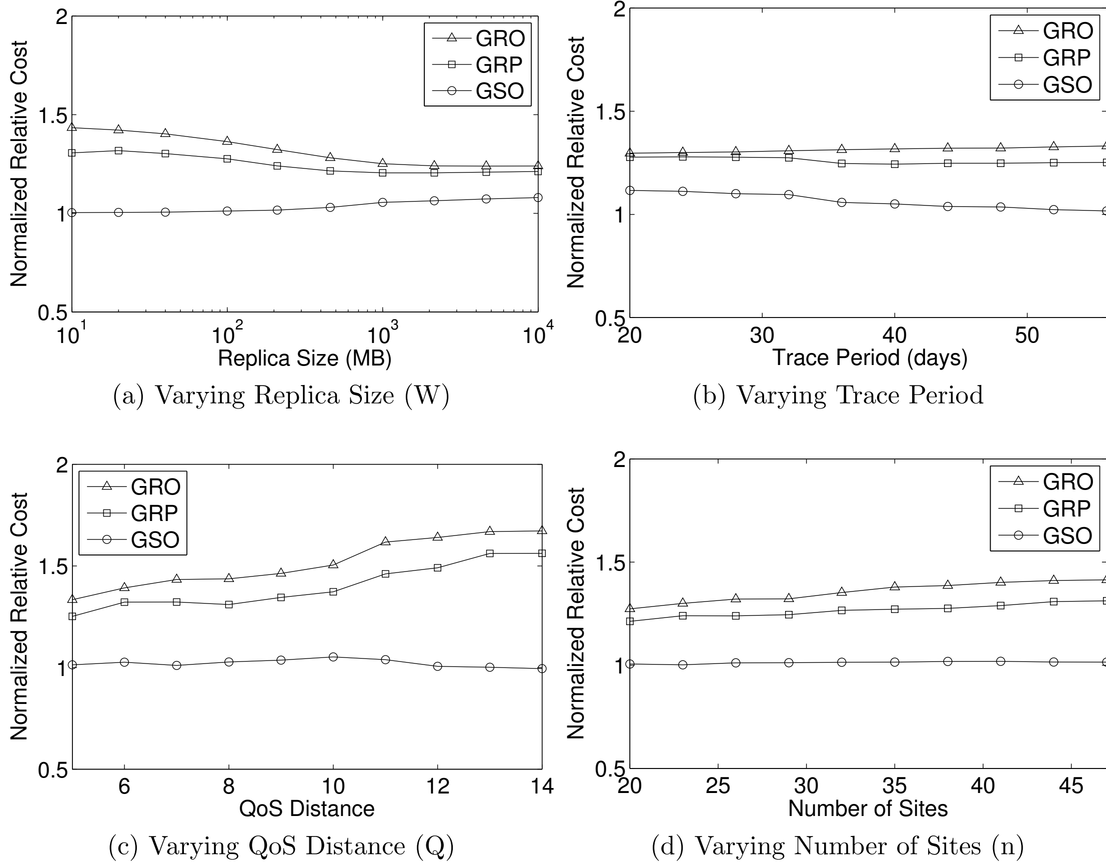

(a) Performance CDF (b) Performance Errorbars

Figure 4.6: QoS Violation Ratios Statistics for Online Algorithms

Nevertheless, the results indicate that past request trace offers a good prediction for future request pattern of the same length in time. In addition, as we use the best offline algorithm Greedy Site (GS) to select sites for preallocation in GRP, GRP has a slight performance advantage over GRO.

It is interesting to see in Figure 4.7 that cost for the static GSO varies differently than the dynamic heuristics (GRO and GRP). In fact, GSO should perform similarly as GS does in offline settings, as it runs GS on past user requests; the other two should perform similarly as GR does, because they both run GR dynamically, regardless of whether they use preallocation or not.

Besides the performance similarity between each heuristic and the offline algorithm it is based on, we have other observations as follows. Static allocation does not incur any cost for opening new sites after the preallocation phase. As a result, the longer time we run a test, the more advantage GSO has by assigning QoS




(a) Varying Replica Size (W)

(b) Varying Trace Period

(c) Varying QoS Distance (Q)

(d) Varying Number of Sites (n)

Figure 4.7: Normalized Relative Cost for Online Algorithms

violated requests to sites already open; thus we see a decrease in the normalized relative cost when we increase the trace period (Figure 4.7b). With increasing QoS distance (Figure 4.7c), the problem space becomes larger and therefore the relative cost is supposed to increase. However, the cost of GSO increases slightly and then decreases. The reason is that GSO can cover more user requests in the preallocation phase with a larger QoS value. In other words, a large portion of the online problem is solved by a offline heuristic. In an extreme case that the QoS value is large enough so that every site covers all users, GSO will have similar performance as the offline GS heuristic.

To summarize, GRP incurs the smallest QoS violation while achieving good performance in terms of cost. GRO incurs slightly more violations but has the advantage of running without past traces. GSO trades a much higher violation

ratio for lower cost and may cause unfairness among users. We suggest the following guideline for using online schemes: use Greedy Request with Preallocation (GRP) as long as there is some past request pattern; otherwise use Greedy Request Only (GRO) which does not rely on the past request pattern. Only when QoS violation is not a large concern should we choose Greedy Site Only (GSO).

## 4.7 Conclusion and Future Work

In this chapter, we investigated the problem of placing Web server replicas in storage cloud-based CDNs along with building distribution paths among them to minimize the cost incurred on the CDN providers while satisfying QoS requirements for user requests. We formulated the problem as an Integer Program and presented various offline and online greedy heuristics. We evaluated their performance against the optimal via Web trace-based simulations.

In the future, we plan to expand our formulation to explicitly consider both bandwidth and storage capacity at a cloud site. In reality, latency can vary because of congestion, network failures, route changes, etc., which is not explicitly addressed in this chapter. We plan to investigate mechanisms to deal with latency variability as well as other aspects of the SLA. For certain web content, for example video, SLA metrics such as throughput are more important.

Chapter 5

# Convergecast and Aggregation in Sensor Networks

In this chapter, we study a scenario in which information flows in the opposite direction. Convergecast is a Many-to-One communication paradigm, in which data from multiple sources are collected, forwarded and aggregated en-route until they reach the base station. In this chapter we focus on wireless sensor networks. We study two TDMA scheduling problems in this setting—maximizing data delivered within a deadline and minimizing the time for delivering all data.

## 5.1 Introduction

Data gathering or *convergecast* in wireless sensor networks is a many-to-one communication pattern in which sensors send data to a base station or *sink*. Each sensor node operates as a data source and/or relay node to pass along data items. A routing tree, specifying paths from all source nodes to the sink, is derived from the connectivity graph of the underlying network. Data at each sensor node is forwarded on to the parent within the tree, until reaching the root, i.e., the sink.

Due to the broadcast nature of wireless communication, interference may occur when multiple sensors transmit data simultaneously, causing the interfering packets to fail. The links of the routing tree (which is given) should be scheduled to fire as efficiently as possible, while observing interference constraints (Two types commonly considered, defined in Section 5.3, are called *primary* and *secondary*.)

When demand for data is high relative to available bandwidth, or time is limited, scheduling transmissions can be a difficult algorithmic problem. Optimization objectives here include minimizing the delivery time for all data or maximizing total data delivered before a deadline [77].

*One-shot* convergecast applies in urgent situations, when a one-time query for data is sent to the sensor network. Each sensor reports a single reading that must be scheduled for delivery to the sink, prior to some deadline. Interference constraints and bandwidth limitations may prevent all data items from being delivered on time. The goal in one-shot throughput maximization is to find a schedule delivering on time as many data items as possible (or more generally a maximum-weight set).

If sensors continuously report data over time, then a periodic schedule is appropriate. Following an initialization period in which the sensors' first data items are delivered, data items will have been received from all sensors. Therefore the natural goal is to minimize the latency of the items sent, and hence maximize throughput. This yields the problem of finding a minimum-length periodic schedule (in which every required transmission fires once). Because of the repetitive nature of a periodic schedule, it is standard to assume that edges on a path may fire out of order, since data can be pipelined between rounds of the schedule. This justification does not apply to one-shot scheduling, where an item's sequence of edge firings towards the sink must occur in consistent order.

Data *aggregation* in wireless sensor networks combines data items from multiple sensors together, en-route, as they travel towards the sink. This reduces the number of transmissions needed, thus saving energy and limiting interference. In certain applications, it is possible to aggregate an arbitrary number of unit-size data items into a single representative data item of the same size. Important examples of this kind include algebraic aggregation functions [78] such as sum, average, min, and max. For example, when monitoring the average temperature of a certain region, the information from all sensors can naturally be merged into a single representative value. In other applications, data collected by different sensors are unrelated and cannot be aggregated, and so all the original, *raw data* items themselves must be delivered. These two settings have been extensively studied, but with aggregation typically paired with periodic scheduling and raw-data paired

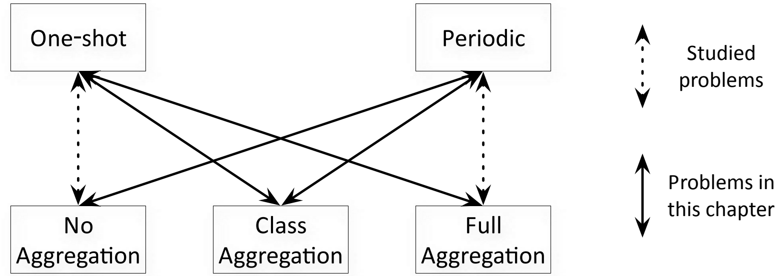



Figure 5.1: Problem relations and combinations.

with one-shot. One intermediate setting that has been proposed is *geographically bounded* aggregation [79] in which data items are aggregatable if they can reach a common ancestor node in the routing tree in limited time.

In this chapter we first study two novel intermediate settings: 1) one-shot scheduling with full aggregation and 2) periodic scheduling with raw data. On the one hand, the one-shot data of homogeneous sensors can be aggregated while waiting for busy links, especially in the case of simple aggregation functions of the kind mentioned above, thus increasing information throughput to the sink. On the other hand, for periodic scheduling, while a sensor may produce data of a single type, such as temperature or acoustic, intermediate relay nodes may be shared and thus multiplex between multiple different types. In the extreme case of this kind, the sensors may all stream raw data.

Later, we generalize to a setting where a data item may potentially be aggregatable with some items but not with others, giving rise to a rich family of problems. In this setting, data are divided into multiple data *classes*. Each class may correspond to semantically similar data items, e.g., temperatures from different nearby locations to be averaged for increased accuracy or multiple counters to be summed together. Only data items within a class are aggregatable. Then *full aggregation* is the special case with one class containing all data items, and raw-data or *no aggregation* is the special case with $n$ (singleton) classes. These are both interesting special cases of the general setting, and several of our results concern them. The relationships among the problem settings and aggregation types are shown in Fig. 5.1.

In summary, our contributions are as follows. We prove that the most general

forms of the two problems are NP-hard and thus provide heuristics for solving them. In our simulation results, we observe that they continue to exhibit good performance. Then we find various restricted settings that admit guaranteed approximations or even optimal solutions, from which we highlight the following.

- **One-shot**: We show the problem is NP-hard to approximate even on trees with primary and secondary interference, strengthening the hardness results of [80]. In special cases where data have a common deadline and secondary interference can be avoided (see the discussion in Section 5.3), we provide a *local* $(hm + 1)$-approximation algorithm (equivalent to a heuristic given without analysis in [80]) and a $2h$-1-approximation algorithm for the no-aggregation setting, where $h$ is the height of the routing tree and $m$ is the number of data items.
- **Periodic**: We show the problem is NP-hard (and NP-hard to approximate with factor better than 4/3) and give the following positive results. We give a constant-approximation algorithm for settings with interference range greater than transmission range, whose factor depends on the ratio of the ranges and is 6 whenever the ratio is at least 4.05. When multiple frequencies can be used, we prove that the full-aggregation BFS algorithm of [78] remains optimal when extended to the setting of data classes and also applies to any ordering that adds leaf nodes iteratively.

The rest of this chapter is organized as follows. Section 5.2 reviews related work. Section 5.3 discusses the system and interference models. Section 5.4 defines the one-shot problem, proves hardness, and presents and analyzes algorithms. Section 5.5 does the same for the periodic problem. Section 5.6 presents experimental evaluation, and Section 5.7 concludes the paper.

## 5.2 Related Work

There is a large literature on link scheduling [81, 82] in general TDMA systems. Unlike our setting, however, the traffic pattern in that work is not many-to-one, and their maximized throughput is generally neither weighted nor end-to-end.

For the many-to-one setting, One-shot scheduling is typically paired with no aggregation while Periodic scheduling is paired with full aggregation. To our best

knowledge, none of the existing work has studied the more general case in which several data classes are delivered and only data items within a class are aggregatable.

One-shot raw data gathering has been studied by e.g. [83, 84, 85, 86]. Gandham et al. [83] considered a single channel with a TDMA protocol and proposed a distributed algorithm to minimize the schedule length. More recently, Ergen et al. [84] proposed a hierarchical scheduling algorithm that schedules the levels of the routing tree before scheduling the nodes. Choi et al. [85] studied settings with simpler interference models, in which similar problems were optimally solvable on lines and trees. Beside introducing aggregation to the problem, our work differs from this literature in that we distinguish between different data items by their weights, and also in that we maximize throughput, given deadline constraints.

Li et al. [80] proved an NP-hardness result, for an arguably artificial problem setting in which *any* conflict graph is possible, and they gave a heuristic for a version of the one-shot problem with no aggregation and non-unit-size data items. We give a stronger hardness result, for the specific case of certain more realistic interference models, and we give an approximation guarantee for their heuristic restricted to unit-size data items.

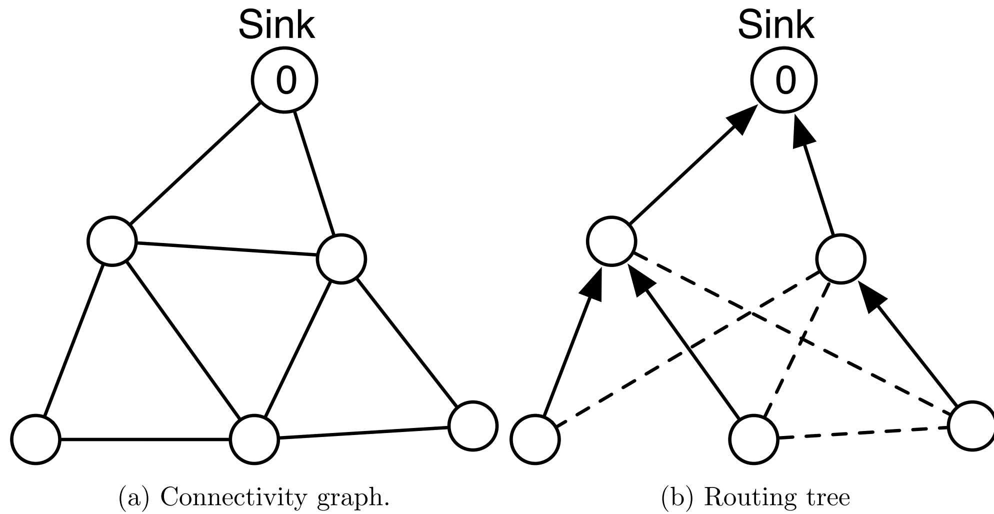
Aggregated convergecast, in which data are combined en-route as they travel to the sink, has been studied by various researchers. Yu et al. [87] gave a distributed algorithm to find collision-free schedules with the latency bound of $24D + 6\Delta + 16$, where $D$ is the network diameter and $\Delta$ is the maximum node degree, later improved on by Xu et al. [88] with a bound of $16R+\Delta-14$ where $R$ is the network radius. Ghosh et al. [78] studied the joint problem of scheduling transmissions and assigning frequencies to the transmissions; they provide optimal solutions in the case in which secondary interference can be eliminated.

## 5.3 Network Model

A group of sensors with wireless transceivers are deployed in a certain region. They form a multi-hop wireless network (see Fig. 5.2a) to forward data to the sink. Like much of the work in this area, we use TDMA [78, 83, 84, 87, 88] rather than contention-based mechanisms for media access control, in order to

avoid interference and provide delivery time guarantees. We assume that each time slot of a link provides enough time for the transmission of exactly one data item. Data of the same type may be aggregated at any sensor node. The resulting data item is also unit-size.

A routing tree for this network (see Fig. 5.2b) is taken as given, constructed by e.g. a centralized mechanism such as Breadth First Search, or a distributed means such as Directed Diffusion [89].




(a) Connectivity graph. (b) Routing tree

Figure 5.2: An example scenario.

We use the graph-based *protocol* model for *primary* and *secondary* interference[1]. In the routing tree, two adjacent edges interfere with one another by primary interference when they transmit at the same time; secondary interference occurs when one receiver is in the (interference) range of a second sender. In the example of Fig. 5.2b, node pairs experiencing secondary interference are connected by dashed lines. We assume the interference range $IR$ is greater than or equal to the transmission range $TR$.

In a centralized setting, the schedule is calculated by the base station and broadcast to the sensors using a preallocated channel, as in wireless-HART [91], in which radio resources are allocated and managed by a centralized unit called

[1]The *physical* interference model [90] is more accurate, but the protocol model is commonly used for purposes of algorithmic analysis.

the network manager; in a distributed setting, the sensors may work cooperatively and construct the schedule themselves.

## 5.4 One-shot Max-throughput

In this section, we first formulate the throughput maximization problem, which we prove to be NP-hard. Then we present and analyze algorithms, which provide guaranteed approximations in certain settings.

### 5.4.1 Problem Formulation

The sensor network is modeled as an undirected graph $G = (V, E)$, where $v_0$ is a *sink* node and $V - \{v_0\}$ is a set of *sensor* nodes. An edge $e = (u, v) \in E$ exists if the two nodes $u, v \in V$ are within communication range of each other.

Let $D = \{d_i : i = 1 \dots m\}$ denote the set of data items. Each $d_i$ is associated with a weight $w_i$ and a source sensor $s_i \in V - \{v_0\}$. Disjoint subsets $b_1, b_2, ...b_l \subseteq D$ are called *data classes*. Each class $b_k$ has a deadline $T_k$, which is the time by which items in $b_k$ must be delivered in order to receive credit for them. Since classes are disjoint, each item's deadline is the deadline of its class. An important special case is when all deadlines are the same.

A routing tree $R$ is given, which spans $G$ and is rooted at $v_0$. For each sensor node $v_j$, let $e_j$ be the corresponding edge connecting $v_j$ to its parent. If node $v_j$ lies on the path that item $d_i$ takes to the root, then let $c(i, j)$ denote the index of the child node of $v_j$ lying on this path. Let $j \sim j'$ indicate that *edges* $e_j, e_{j'}$ interfere (either through primary or perhaps secondary interference), and let $i \sim i'$ indicate that *data items* $d_i, d_{i'}$ are of *different* classes.

We formulate this problem in IP (5.1). Decision variable $x_{ijt} = 1$ indicates that item $d_i$ is transmitted on link $e_j$ (from $v_j$) at time $t$. The objective is to maximize the weighted number of data items that reach the sink before their deadlines. The first constraint set indicates that for each data $d_i$, if the transmitting node $v_j$ is not the source, the child of $v_j$ must have transmitted $d_i$ to $v_j$ before the current time $t$. The second constraint set ensures that no interfering nodes will transmit at the same time. We also implement aggregation implicitly in this set of constraints, by

allowing items from the same class to fire simultaneously on the same link, which corresponds to transmitting the aggregated item. The third constraint set prevents any data item from being delivered more than once.

$$
\begin{aligned}
\max \quad & \sum_{i,t \leq T_i} w_i x_{ic(i,0)t} & & \quad (5.1)\\
\text{s.t.} \quad & \sum_{t' < t} x_{ic(i,j)t'} \geq x_{ijt} & & \forall i, j, t \\
& \sum_{i' \sim i} x_{i'jt} + \sum_{i', j' \sim j} x_{i'j't} \leq 2m(1 - x_{ijt}) & & \forall i, j, t \\
& \sum_{t \leq T_i} x_{ic(i,0)t} \leq 1 & & \forall i \\
& x_{ijt} \in \{0, 1\}
\end{aligned}
$$

### 5.4.2 Hardness

In [80] it was shown that the one-shot throughput problem was NP-hard when the conflict graph between data items could be completely arbitrary. In fact, we show that the problem already becomes hard with realistic interference models.

**Theorem 1.** *With primary and secondary interference, the one-shot throughput problem is NP-hard to approximate, even in the special cases of unit weights ($w_i = 1$) with 1) full aggregation and 2) no aggregation.*

*Proof.* Given is an instance of the Maximum Set Packing problem with a universe $U$ of $n$ elements and a family of $m$ subsets, which is NP-hard to approximate with factor $n^{1-\epsilon}$ [92]. The task is to choose a maximum number of disjoint subsets. To construct a one-shot problem instance, for each subset $S_i$, we introduce a path to the sink with a data item at the leaf (see Fig. 5.3). The paths share no vertices until the root, so there is no opportunity for aggregation and the result will apply to any number of classes, including 1 and $m$. All paths are of length $\ell$.

The paths progress from left to right, smoothly zig-zagging up and down, in and out of what we call *element regions*, each of which will correspond to an element $e_j$ of $U$ and is of a fixed length $r$. Paths in the element regions are drawn sufficiently tightly allowing only one path to progress at a time; and paths in non-element regions are drawn sufficiently far apart, so that they cause no

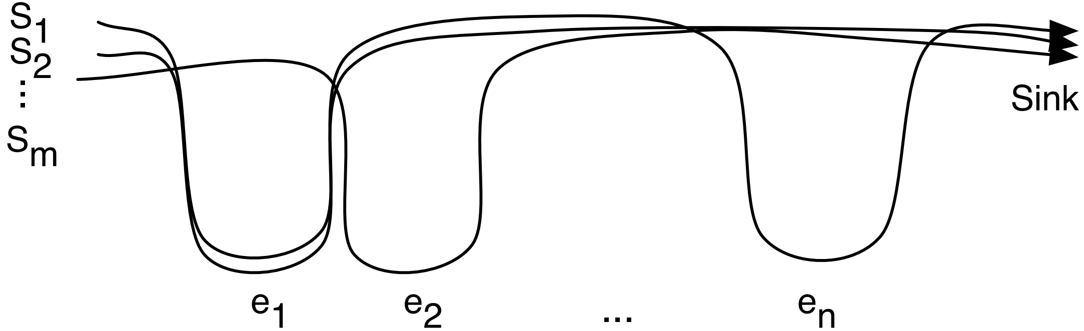



Figure 5.3: Hardness reduction for one-shot with secondary interference.

secondary interference. By carefully choosing $\ell$ and $r$, we can ensure that any data item delivered on time prevents the on-time delivery of other data items it shares element regions with. Each such choice corresponds to the selection of one set precluding the selection of sets sharing elements with it. Hence any valid set packing solution can be converted to a corresponding one-shot throughput solution of the same size, and vice versa. □

### 5.4.3 Heuristics for the General Case

We now present two heuristics and discuss their approximation guarantees under different settings.

A common subroutine *Is-Schedulable* is used to check if a data $i$ is allowed to transmit on node $j$. It returns false if an interfering edge is transmitting simultaneously or if the present edge is transmitting an item from another class (hence not aggregatable). As in the IP formulation, the subroutine implicitly performs *opportunistic* aggregation, by allowing multiple data items of the same class to transmit over the same link simultaneously, if they happen to be then present. An alternative is *mandatory* aggregation, in which we would hold data items at a given node until items of their class within the subtree have arrived there. That is, same-class items are aggregated at their lowest common ancestor in the routing tree. We found opportunistic to usually outperform mandatory in our experiments, and so we focus on opportunistic aggregation in this chapter. These algorithms also use a *sort* subroutine which may sort data items by any combination of the following criteria: 1. decreasing weight, 2. increasing last starting time (LST, after

when a data must be late), 3. increasing distance to the sink.

#### 5.4.3.1 MIS-based Algorithm

The first algorithm (see Algorithm 12) makes decisions over time, looking for a maximal concurrent transmission set of data items in each time slot. To find this concurrent set, we run a greedy heuristic for the Maximal Independent Set (MIS) problem. That is, in each time slot, after sorting data items, we check and schedule every data item by the sorted order.

**Algorithm 12** MIS-based Algorithm

```
1: for each timeslot do
2:    scheduled link set S ← ∅
3:    sort data items
4:    for each d_i (located at v_j) do
5:       if Is-Schedulable(d_i, v_j) then
6:          S ← S ∪ e_j
7:       end if
8:    end for
9:    remove any items impossible to deliver on time
10: end for
```

This algorithm may also be implemented in a semi-distributed way, using local knowledge. In each timeslot, each node's best data item will compete with those of other interfering nodes for transmission in that timeslot, corresponding to a selection of nodes in a MIS, which can be found using a distributed MIS algorithm (e.g., [93, 94]).

In [80], an algorithm was presented in which jobs (data transmissions) are assigned to timeslots by First-Fit, choosing jobs in order of Last Starting Time (LST), with the proviso that once it is no longer possible for a data item to reach the sink, jobs corresponding to it are removed from consideration. In the case of unit-size data items, this algorithm is equivalent to our MIS-based algorithm when data items are sorted by LST.

The algorithm clearly provides no approximation guarantee in the case of weighted data items, so we will analyze it in the case of unit-weight items. In fact, even with primary interference only, a common deadline for all items, and no aggregation, the approximation ratio is unbounded and can be as bad as $m/2$

(where $m$ is the number of items). An example problem instance is shown in Fig. 5.4 in which each leaf node contains an item with deadline $m$. The algorithm will forward $m-1$ of the items one hop and then deliver just one of them; the optimal solution delivers $m/2$ items, for a ratio of $m/2$.

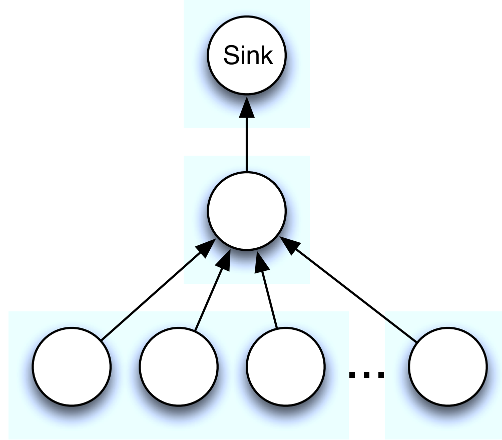



Figure 5.4: Bad example for MIS-based algorithm with LST.

The following result upper-bounds the approximation ratio.

**Theorem 2.** *Let $h$ be the height of the routing tree and $m$ be the number of items. Then in the no-aggregation setting with primary interference only and a common deadline, the MIS-based algorithm provides a $hm+1$-approximation.*

*Proof.* Let the *level* of a node be its distance from the root, let the height of the tree be the maximum level of any node, and let $d_{min}$ be the minimum level of any node where an item is initially located.

First observe that only deadlines greater than or equal to $d_{min}$ is non-trivial, since otherwise nothing can be delivered.

Let the deadline be some time $T$. First, suppose it is at most $hm+1$. Then we claim that at least one item must be delivered. To see this, notice that in the worst case, during the first $m$ timeslots only leaf nodes fire. Then after beginning at time $m+1$, some item must fire from level at most $h-1$. More generally, at time slot $im+1$, some item must fire from level at most $h-i$, and so by time slot $hm+1$, some item must reach the root.

For the remainder of this proof, let a *round* mean a sequence of $hm+1$ time slots.

If $T$ is greater than $hm + 1$, then we know that at least one item is delivered during the first round. During the second round, at least one more item is delivered, by the same argument, and so on. If we run for $k$ full rounds and $T = k \cdot (hm + 1) + t_0$, then at least $k$ items are delivered. Now, consider the last $t_0$ time slots. Recall that in the worst case of delivering a single item in one round, at the end of the round, all but one of the root's children has an item. Eventually, at the last time slot, all items will be dead except those at the root's children, and so one of these must then fire. Therefore with a deadline of $T = k \cdot (hm + 1) + t_0$, we necessarily deliver at least $k + 1$ items. OPT is always upper-bounded by the number of time slots, and so the result follows. □

#### 5.4.3.2 Straight-through Greedy (STG)

The second algorithm (see Algorithm 13) works item-by-item. Instead of scheduling all the transmissions for each timeslot, it looks for a sequence of transmissions to transmit the current data item all the way from source to sink, thus avoiding transmissions for data items that eventually fail. Data items are attempted in *sort* order.

**Algorithm 13** Straight-through Greedy (STG)

1: *sort* data items
2: **for** each $d_i$ **do**
3:    $d_i$ is at $v_j$
4:    **while** $d_i$ is able to make the deadline **do**
5:       find the smallest $t$ when *Is-Schedulable*$(d_i, v_j)$
6:       schedule $e_j$ with $d_i$ at $t$
7:       $j \leftarrow p(j)$
8:    **end while**
9:    **if** $d_i$ did not reach the sink **then**
10:       remove all occurrences of $d_i$ from the schedule
11:    **end if**
12: **end for**

In a similar setting as to the one in which the MIS-based algorithm provides a $hm + 1$-approximation, STG gives a better performance guarantee.

**Theorem 3.** *With no aggregation and primary interference only, Straight-through Greedy provides a* $2h - 1$*-approximation, where* $h$ *is the height of the routing tree.*

*Proof.* We prove by induction on the number of items.

Assume unsorted STG provides a $2h-1$-approximation on all problem instances with at most $k$ items. Now consider an instance $I$ consisting of a time-index graph and $k+1$ items (with current positions at time 1). Let $i$ be the first item chosen by the algorithm, and let $p$ be the time-indexed path chosen for $i$. Let $I'$ be a problem instance the same as $I$ except with item $i$ removed and all time-indexed edges either appearing in or conflicting with those appearing in $p$ removed. By the inductive hypothesis, we have $ALG(I') \geq 1/(2h-1)OPT(I')$.

We now argue that $OPT(I) - OPT(I') \leq 2h-1$, i.e., that the removal of item $i$ and the interference caused by the path $p$ reduces the value of $OPT$ by at most $2h-1$. There are three ways an item $j$'s delivery could depend on these edges. Consider an edge $e_i \in p$ that $j$ depends on and an edge $e_j$ that $i$ depends on. First, it could happen that $e_i = e_j$ or that $e$ and $e'$ share the same receive nodes (Fig. 5.5a). Second, it could happen that $e_i$'s receive node is $e_j$'s send node (Fig. 5.5b). Third, it could happen that $e_i$'s send node is $e_j$'s receive node (Fig. 5.5c). The path $p$ can contain at most $h$ edges. Say that an edge disrupts another item if its firing prevents the item's delivery, due to one of the three types of interference. Each of these edges except for the last could disrupt a type-two interference or disrupt two items simultaneously, due to types one and three; the last could disrupt at most one other item that would otherwise be delivered, for a total of $2h-2$ other items disrupted, or $2h-1$ including item $i$. Then we have:

$$\begin{aligned} ALG(I) &= 1 + ALG(I') \\ &\geq \frac{1}{2h-1}(OPT(I) - OPT(I')) + \frac{1}{2h-1}OPT(I') \\ &= \frac{1}{2h-1}OPT(I) \end{aligned}$$

□

We found primary-interference problem instances in which the scheduling of one data item blocks, and ultimately prevents delivery of, two data items that could have jointly succeeded, although these instances do not provide tight lower bounds. Consider the example shown in Fig. 5.6. First assume all four items have a deadline equal to the distance of the lower-most item from the root. Then if

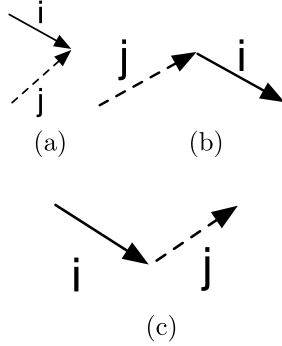



Figure 5.5: Interference patterns.

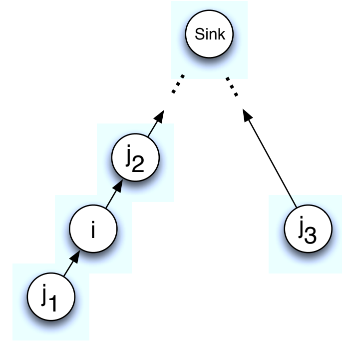



Figure 5.6: Bad example for STG.

item $i$ is chosen first, a solution value of 1 obtains rather than 3. If $i$ has a greater deadline, the factor can increase to 4. In both cases, the choice of $i$ causes at most one instance of each interference pattern.

In the case of aggregation, however, examples can be constructed to show that both two greedy algorithms can do arbitrarily badly, because one data item blocked by our greedy choices could be an aggregated data item which is much more valuable, even in the case of unit-weighted data.

## 5.5 Min-length Periodic Scheduling

We now turn to the periodic setting. First, we formulate the problem as an integer program and address hardness. Then we adopt and extend two existing algorithms.

### 5.5.1 Problem Formulation

The sensor network model is similar to that of Section 5.4.1. Let $b_1, b_2, ..., b_l$ denote the data classes. Given the routing tree $R$, all data sources of $b_i$ together with the nodes connecting these sources to the sink form a subtree $R_i \subseteq R$. For each $b_i$, every edge of $R_i$ must transmit once in each period. That is, an edge of $R$ must fire once per period for each subtree $R_i$ it appears in. Let $L = \{(i,j)|\ e_j \in R_i\}$ be the set of transmissions to be scheduled in periods. Decision variable $x_{kt}$ for $k = (i,j) \in L$ indicates whether transmission $k$ fires at time $t$; $y_t$ indicates whether at least one transmission fires at time $t$.

The problem is formulated in IP (5.2). The objective is to minimize the total

number of timeslots that pass before all requisite transmissions have fired. It is upper-bounded by the total number of transmissions $|L|$. Notice that the objective function sums all variables $y_t$, which is the first set of constraints defined to indicate whether time slot $t$ is used, i.e., any transmission fire at time $t$. If any time slot were not used in an optimal solution, then the schedule could be improved by shifting all subsequent transmissions to one earlier time slot, and so the objective function indeed captures the right optimization goal. The second set of constraints prevents any interfering nodes from transmitting at the same time.

$$\begin{aligned} \min \ & \sum_t y_t & & \quad (5.2) \\ \text{s.t. } & x_{kt} \leq y_t & & \forall\ k \in L, t \in [1, |L|] \\ & x_{kt} + x_{k't} \leq 1 & & \forall \text{ interfering } k, k' \\ & y_t, x_{kt} \in \{0, 1\} & & \end{aligned}$$

### 5.5.2 Hardness

We prove the problem is NP-hard to approximate with factor better than 4/3. To show this, we adapt the reduction of [95] for the problem of coloring a graph of degree 3.

**Theorem 4.** *The periodic problem is NP-hard to approximate with factor 4/3 or better, even in the special case of full aggregation.*

*Proof.* A reduction is given in [95] from 3-coloring a planar graph of degree 3 to 3-coloring a unit disk graph (UDG). The reduction proceeds by replacing each node of the graph with a unit disk and each edge with a chain unit of disks, as shown in Fig. 5.7a. The chains are constructed in such a way that the endpoints of each chain must receive different colors in any 3-coloring. Since the degree of the original graph is 3, each node has up to 3 chains attached. The granularity of the chains (equivalently, the size of the unit disks) can be chosen so that disks from different chains do not overlap.

We now extend the construction as follows, constructing a routing tree with the specified UDG as its conflict graph (see Fig. 5.7b). Since the underlying graph is a tree, we speak in terms of interfering nodes rather than edges. Each disk will

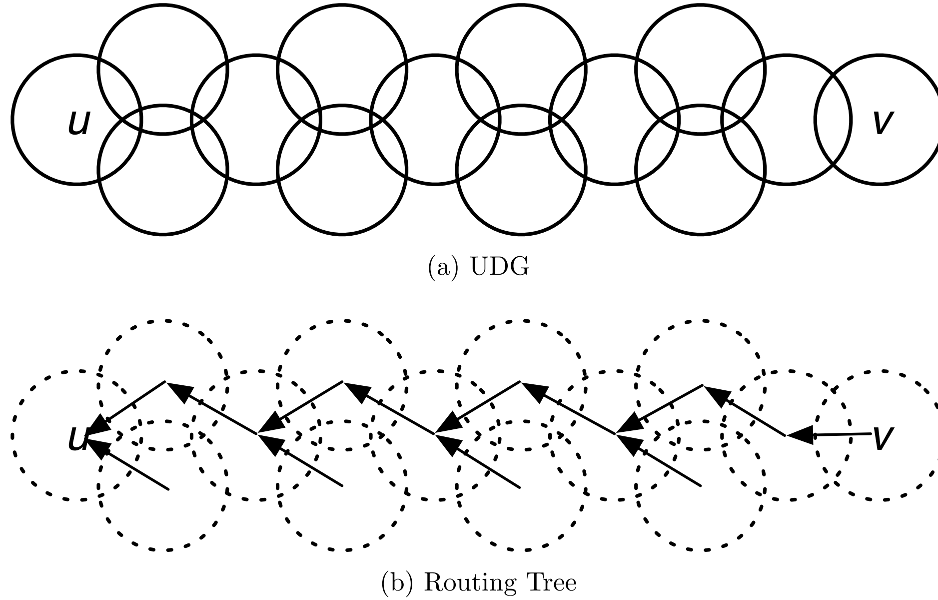



Figure 5.7: NP-hard proof for min-length periodic scheduling.

correspond to a sensor node. We choose an arbitrary node in the UDG to be the sink. Each edge chain $(u, v)$ will be oriented towards the root as follows. Suppose $u$ is the end closer to the root. The chain alternates between disks and disk pairs. For each disk pair, both nodes will forward data to the node to their left; for each single disk, the node will forward data to the upper node of the pair to its left.

The nodes of each disk pair will have primary interference with one another and with their parent, as will each single node with its parent. Finally, consider a single disk $a$ and the lower disk $c$ of the pair $(b, c)$ to its left. Since all three disks intersect, $a$ and $c$ will experience secondary interference. Due to the granularity of the drawing, there is no other secondary interference.

Thus we have constructed a routing tree whose conflict graph is precisely the UDG given, and a periodic schedule of length 3 for it will be equivalent to an edge coloring of size 3 of the UDG. □

### 5.5.3 Algorithms

The algorithms we present involve a common data structure: a directed acyclic multigraph (DAM) or multitree. Vertices in the DAM are the same as in the routing tree $R$. Then for each transmission $k = (i, j) \in L$, we add an edge $e_{ij}$

in the DAM between the end points of $e_j \in R$. As a result of possible multiple transmissions on a node, multiple edges may be added between two nodes; vertices without any edges are eventually excluded from the multitree.

#### 5.5.3.1 Node-based Algorithm

For the general case, with both primary and secondary interferences, Algorithm 14 iterates over the nodes according to a certain order and schedules a node's transmissions by First-Fit. We sort the nodes using one of the following criteria, breaking ties for one ordering using the other:

1. decreasing degree,
2. increasing depth (BFS order).

**Algorithm 14** Node-based Greedy

```
1: sort nodes
2: for each v_j ∈ V_M do
3:    for each item e_ij ∈ E_M do
4:       schedule e_ij at the earliest slot t
5:    end for
6: end for
```

In the case of primary interference only, finding a schedule on a routing tree $R$ or a DAM is the same as coloring its edges. Ghosh et al. [78] showed that in the case of full aggregation and primary interference only, a BFS First-Fit coloring algorithm produces an optimal solution of $\Delta(R)$ colors (where $\Delta$ indicates degree). We now extend this result to multiple classes and to any traversal order that adds a leaf node in each step.

**Theorem 5.** *Assume primary interference only, and let a DAM $R$ representing the routing trees of multiple classes be given. Coloring its edges by First-Fit, with any traversal order adding a leaf node in each step, produces an optimal solution with $\Delta(R)$ colors.*

*Proof.* Build the tree by any traversal order that iteratively adds leaf nodes to the tree. Let $R_i = (V_i, E_i)$ denote the subtree of $R$ after the $i$th iteration. Note

that when the $i$th node connects to the tree, all the edges from it to its parent are colored and added to $E_i$. For $i = 1$, only one node connects to the sink, and so the number of colors used is $\Delta(R_1)$.

Now assume the number of colors used up through the $i$th iteration is $N_i = \Delta(R_i)$. Then in the $i+1$st iteration, there are two possible situations. Let $p_{i+1}$ denote the degree of this $i+1$st node's parent and $q_{i+1}$ denote the number of edges between this node and its parent. The degree of its parent node becomes $p_{i+1} + q_{i+1}$. We do not change any other nodes' degree because this node is leaf node. If $p_{i+1} + q_{i+1} \leq \Delta(R_i)$, then we have enough colors to color these new edges; it also implies that $\Delta(R_{i+1})$ has not increased, and so $N_{i+1} = N_i$. On the other hand, if $p_{i+1} + q_{i+1} > \Delta(R_i)$, then we need to increase the number of colors to $N_{i+1} = p_{i+1} + q_{i+1}$, which is also the new degree of this tree. Either way, we have $N_{i+1} = \Delta R_{i+1}$. □

Similar to the MIS-based algorithm for one-shot problem, this algorithm may also be implemented as in a semi-distributed way, using local information only. Using a distributed sorting or leader election algorithm, we can obtain a sequence of nodes by which the First-Fit scheduling can proceed. Each node then only needs to be kept apprised of its neighbors' scheduling plans. The Node-based algorithm is more intuitive and easier to implement than the following Transmission-based algorithm, but it lacks the flexibility of scheduling each transmission of a node separately.

#### 5.5.3.2 Transmission-based algorithm

Algorithm 15 considers each transmission in the DAM separately. We build a conflict graph where each vertex is a transmission in the DAM, and an edge indicates that the two transmissions conflict with each other. Therefore, finding a schedule in the DAM is the same as coloring vertices in this conflict graph. Note that this applies to the case with both primary and secondary interferences.

To color the conflict graph we use the Inductivity algorithm of Hochbaum [96] (see Algorithm 15), which colors a graph using at most $\delta(G)+1$ colors, where $\delta(G)$ is the largest value $\delta$ such that there exists an induced subgraph $H$ all of whose nodes have degree at least $\delta$. We follow [97] in analyzing the performance of this

algorithm on our interference model. We will show that coloring vertices in the conflict graph by inductivity guarantees a constant-approximation whenever the interference range $IR$ is strictly greater than the transmission range $TR$. We call two edges are *independent* if they are non-interfering.

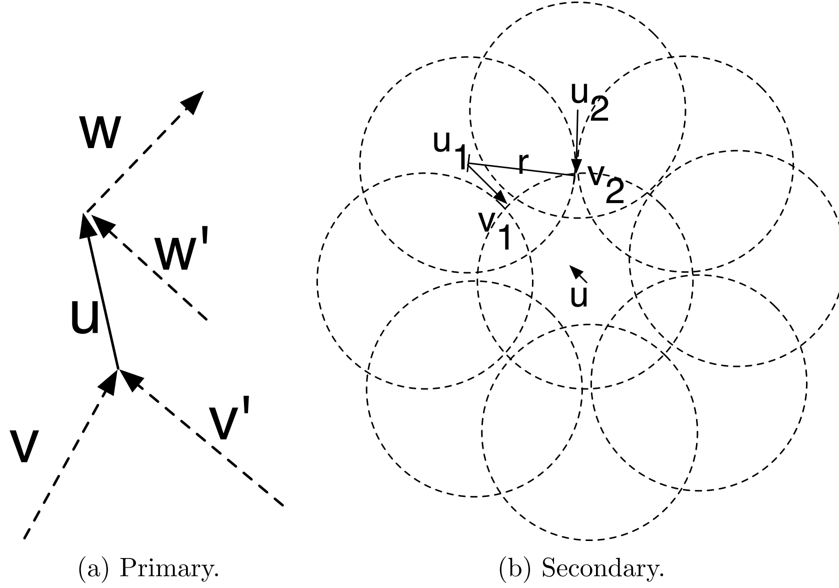
**Algorithm 15** Transmission-based (Inductivity)

1: $n \leftarrow$ the number of vertices in the conflict graph $G$
2: **for** $i \leftarrow 1$ to $n$ **do**
3: find a node $v^*$ of minimum degree in G
4: $v_i \leftarrow v^*$
5: $G \leftarrow G - v^*$
6: **end for**
7: **for** $i \leftarrow n$ to 1 **do**
8: color $v_i$ by First-Fit
9: **end for**

**Lemma 1.** *Let $R$ be the routing tree (or DAM), and let $G = G(R)$ be the conflict graph of $R$. Let $TR = 1$ and $IR = r > 1$. Then the number of mutually independent neighbors a node $u$ in $G$ can have is bounded by a constant.*

*Proof.* Two nodes in $G$ are adjacent if the corresponding nodes in $T$ cannot be scheduled simultaneously, due to either primary or secondary interference. Restricted to primary interference, a node $u$ can be adjacent to at most two independent edges. (In Fig. 5.8a, e.g., transmission on $u$ can only block two independent edges: $v$ or $v'$ and $w$ or $w'$.)

The case of secondary interference is more complicated. Let node $u$ be sending to another node. Suppose $u$ causes secondary interference for nodes $v_1, ..., v_k$, which would be able to receive from other nodes simultaneously. Since each node $v_i$ can receive from at most one node at a time due to primary interference, let $u_i$ be a representative node which sends to $v_i$ in $R$. Let $d(x, y)$ be the distance from $x$ to $y$. Due to $TR$ and $IR$, we have for each $v_i$ that $d(v_i, u) \le r$ and $d(v_i, u_i) \le 1$ (see Fig. 5.8b). In order for none of the $u_i$ nodes to interfere with one another, it must be the case that $d(u_i, v_j) > r$, $\forall i \ne j$. Therefore, there is a disk $D_r(u_i)$ of radius $r$ centered on each $u_i$ in which no other $v_j$ can lie. In the extreme case of maximum interference for $u$, the $u_i$ surround $u$ on a ring, of radius as large as possible given $d(v_i, u) \le r$; the number of $u_i$ is as great as possible, consistent with $d(u_i, v_j) > r$.




(a) Primary.

(b) Secondary.

Figure 5.8: Independent edges blockable by another edge.

Since $|\overrightarrow{uu_i}| = r + 1$, $|\overrightarrow{u_i v_j}| = r$, $|\overrightarrow{uv_j}| = r$, $\angle \overrightarrow{uu_i}\overrightarrow{uv_j} = \arccos(\frac{r+1}{2r})$. In total, there can be as many as

$$\lfloor \frac{2\pi}{\arccos(\frac{r+1}{2r})} \rfloor$$

independent transmissions. Note that no other sender causing primary interference with $u$ can be independent of these $u_i$s. This yields a maximum of 8 independent transmissions, when $r = 2$ or 7 independent transmissions when $r = 3$. □

By an argument similar to Theorem 8 of [97], it now follows from the lemma that:

**Theorem 6.** *The Inductivity algorithm provides a* $\lfloor \frac{2\pi}{\arccos(\frac{r+1}{2r})} \rfloor$*-approximation for the problem of scheduling the edges of* $R$.

The bound function is monotonically decreasing, and

$$\lim_{r \to 1} \frac{2\pi}{\arccos(\frac{r+1}{2r})} = \infty, \ \lim_{r \to \infty} \lfloor \frac{2\pi}{\arccos(\frac{r+1}{2r})} \rfloor = 6$$

When $r = IR/TR = 1$, no guarantee is given. For ratios greater than 1, the bound is never better than 6; however, when $r > 4.05$, it reaches the value of 6.

## 5.6 Evaluation

In this section, we present simulation results evaluating our algorithms. Although our approximation guarantees depend on various assumptions on the problem instances, we test them empirically on general problem instances, i.e., instances with both primary and secondary interference, multiple data classes and (for one-shot) varying data deadlines in order to learn how well the algorithms perform on typical instance where our assumptions may not hold.

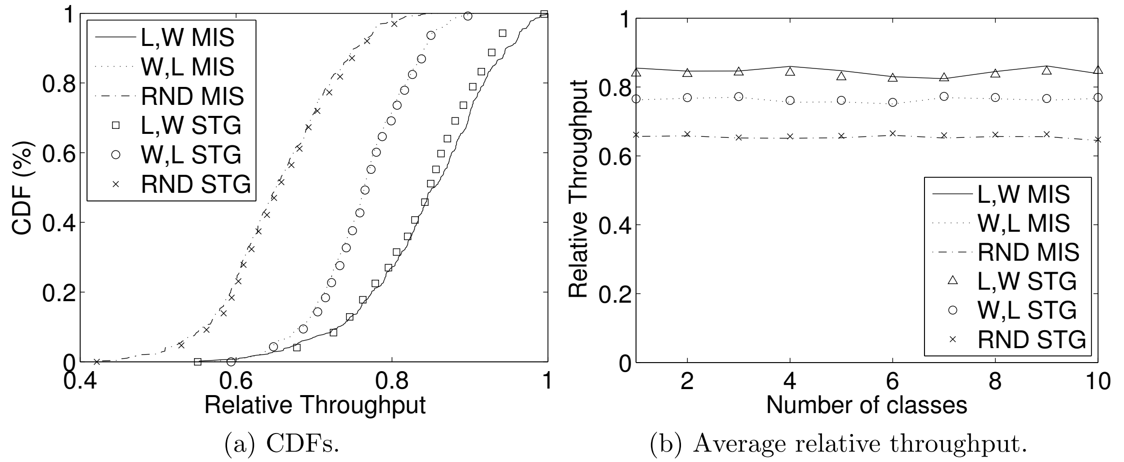
In our simulation setup, 30 sensors with uniform transmission and interference ranges are positioned randomly in the field. The routing tree is built by executing Breadth First Search on the connectivity graph. There are $m$ data items, which are assigned initial positions by choosing nodes uniformly at random, with replacement. Therefore a node may begin with zero, one, or multiple data items.

Each experiment varies one parameter, and is averaged over 50 trials. The solution values plotted are the ratios of algorithm performance to the IP optimal value, which is obtained by the CPLEX solver [47]. These ratios are upper-bounded by 1 for throughput maximization and lower-bounded by 1 for schedule length minimization.

### 5.6.1 One-shot Scheduling

For the one-shot algorithms, there are 30 data items with weights uniformly distributed in $[1, 30]$. Data deadlines are chosen uniformly at random in $[1, 20]$. We vary the number of data classes from 1 to 10, and randomly distribute the 30 items among these classes. We tested both MIS-based and STG with various sorting criteria, including Weight (W), Distance (D) and LST (L) and found that Distance consistently performed the worst. Therefore, in the result shown below, we only plot the following combinations of the sorting criteria for both MIS-based and STG:

- increasing LST, decreasing weight $(L, W)$
- decreasing weight, increasing LST (*W,L*)
- random order (*RND*)




(a) CDFs.

(b) Average relative throughput.

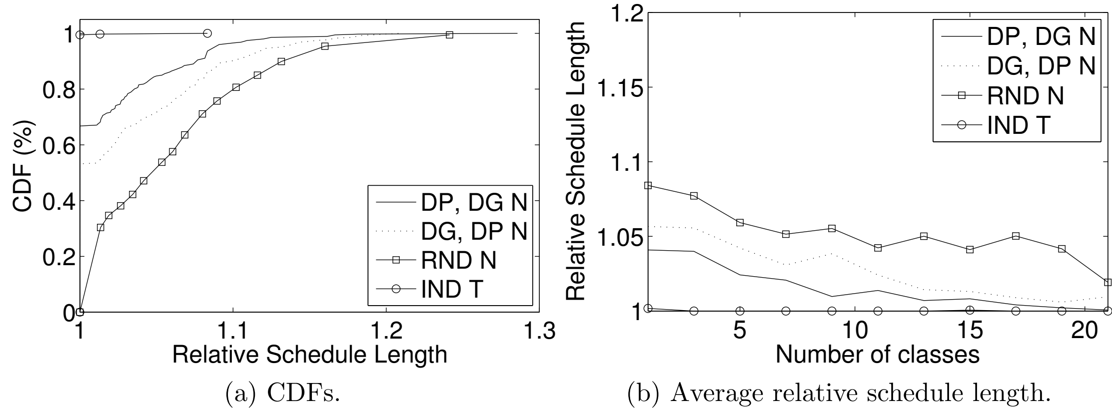
Figure 5.9: Evaluating one-shot convergecast algorithms.

With two criteria we sort by the first criterion and break ties with the second; in the case that there is still a tie, we break it with Distance by picking data closer to the sink.

First we plot a cumulative distribution function (CDF) for each algorithm in Fig. 5.9a, which shows that the MIS-based and the STG performed similarly when the same sorting criteria are used. The deciding factor on performance is the sorting criteria combination. We find the best combination to be $L, W$, which outperforms the rest on every percentile. The MIS-based algorithm with this combination achieves results better than 0.85 of the optimal, 50% of the time.

To investigate the effect of the number of classes on performance, we plot the the average relative throughput against the number of classes in Fig. 5.9b. Perhaps surprisingly, the average throughput does not change significantly as the number of classes changes. This suggests that the opportunistic aggregation works well with various number of data classes and the performance of the heuristics is stable.

Random order performed less well. However, it not only provides a baseline result for comparison purposes but also can be used as a simple distributed algorithm. With an algorithm in which every node competes for its own timeslots, transmission order will depend on the order in which nodes attempt to send, which may reasonably be treated as random.




(a) CDFs.

(b) Average relative schedule length.

Figure 5.10: Evaluating periodic convergecast algorithms.

### 5.6.2 Periodic Scheduling

For the periodic problem we conduct similar experiments, testing the Transmission-based (T) algorithm with Inductivity (IND), and Node-based (N) algorithm with the following sorting criteria:

- increasing depth, decreasing degree (DP, DG)
- decreasing degree, increasing depth (DG, DP)
- random order (RND)

In the CDF plots (Fig. 5.10a),we can see that Transmission-based with Inductivity almost always obtains the optimal solution. Its empirical worst case length is only 1.1 times the optimal. One reason it performs so well may be that it works directly on the conflict graph, in which the competing entities are transmissions rather than nodes. This algorithm may be harder to implement in a distributed way, however.

The best sorting criteria combination for the Node-based algorithm is *DP, DG*, which achieves the optimal in more than 35% of the time and has an empirical worst case of 1.3 times the optimal.

All algorithms performed well, compared to the results for the one-shot problem, including even the baseline random order. We conducted a second series of tests with 100 nodes and 100 data items and found that the relative schedule

length did not significantly increase. This better approximation performance of the periodic algorithms over the one-shot algorithms' is consistent with the hardness of approximation results given in the preceding sections: the hardness of approximation result for the one-shot problem was much stronger than that for the periodic problem. Since the periodic problem appears easier to approximate in practice, the Node-based algorithm might be preferred, especially if a distributed implementation in required.

In addition, we plot the average relative schedule length as a function of number of classes in Fig. 5.10b. It is interesting to see that the relative schedule length varies inversely (except for Transmission-based) with number of classes. With a fixed number of data items, more data classes implies that there are more edges in the DAM that must fire in a period. But the problem is not harder to solve by Node-based, because we have the same number of nodes to sort. In fact, with fewer data classes, the optimal schedule length is small. Therefore, the price of being non-optimal is high with fewer classes, as we can see a higher relative schedule length in Fig. 5.10b.

## 5.7 Conclusion

Table 5.1 summarizes the algorithmic results (the columns indicate the interference model used (primary only or primary and secondary), the aggregation setting ($c$ = the general aggregation with multiple classes, 1 = full aggregation, $n$ = no aggregation), whether the deadlines are common or not, and the corresponding approximation guarantee obtained).

| problem | algorithm | IF | CL | DL | approx. |
|---|---|---|---|---|---|
| One-Shot | MIS-based | P&S | n | same | $mh+1$ |
| | STG | P | n | same | $2h-1$ |
| Periodic | Node-based | P | c | n/a | optimal |
| | Inductivity | P&S | c | n/a | $\lfloor \frac{2\pi}{\arccos(\frac{r+1}{2r})} \rfloor$ |

Table 5.1: Summary of algorithmic results.

In the future we will explore more combinations of interference models and

aggregation types, seeking less special cases of the problems that admit guaranteed approximation or optimal solution. We will also examine settings in which data items in a class have a nonadditive joint utility.

Chapter 6

# Stochastic Resource Allocation

In previous chapters, resource or information demands are either static and known a priori in offline settings, or fixed once revealed in online settings. In this chapter, we study an extension to the previous chapters, in which demands in resources are stochastic and change over time. We model this problem as a stochastic multi-dimensional knapsack problem. Instead of strictly enforcing the knapsack constraints, we allow occasional violations within an overflow probability.

## 6.1 Introduction

Networked applications are growing in popularity and complexity. They are supported by a variety of end devices, services in the cloud and different types of networks. Resources in modern computer and communication systems include those in end devices such as CPU, memory and disk space, and in networks such as bandwidth and spectrum. To serve a user running networked applications, we need to satisfy resource requirements in multiple dimensions within the capacity constraint of each dimension. Therefore, one computing system, as the unit with which some limited resources are associated, can only admit a subset of the users/tasks at the same time. If we assign weight or profit to tasks, our goal is to maximize the total profit admitted to the system without exceeding the resource limitations.

If tasks have deterministic demands, think of the capacity of each resource as the size of a knapsack in one dimension; we are assigning tasks to a knapsack

with respect to multi-dimensional capacity constraints. This is a multi-dimensional knapsack problem. In practice, however, demands for resources may change over time and only their statistics or distributions are known a priori. In other words, the demand of each task is a random vector, the elements of which indicate its varying needs for multiple resources.

One simple case is that the demand vectors follow a simple distribution, such as a Bernoulli distribution of $(s, q)$. That is, at each step, a mission either requires $s$ of a resource with the probability of $q$, or none of the resource. Other examples may be an exponential distribution (implying a certain service time), a Poisson distribution (indicating an arrival rate of event) or any combinations of these distributions in different dimensions.

A quick solution would be assigning tasks based on their maximum possible demands. This not only leads to low utilization of resources but also applies to few distributions; tasks with unbounded distributions such as the exponential distribution may never be admitted to the system. *Statistical multiplexing* [98] is more suitable in that uncorrelated demands are packed together allowing occasional violations of the capacity constraints. Particularly, the *overflow probability*, which indicates the chance that admitted tasks violate the capacity constraints, should be bounded by a predetermined value $p$.

In general, solving such a problem relates to solving a chance-constrained program, which involves various non-linear optimization techniques. Recently, the Sample Average Approximation (SAA) method has been used in solving such a problem by Luedtke and Ahmed [99] and Pagnoncelli *et al.* [100]. The authors provide promising results for a more general problem but the solution is based on Monte-Carlo simulation. We use the SAA method only to derive an upper bound and evaluate our algorithms; our scope is to find efficient heuristics for this particular problem.

The main idea in solving this problem lies in the development of an *effective bandwidth* or *effective size* for stochastic demands. The concept of effective bandwidth has been widely used in the field of admission control and bandwidth management. This is a special case of our problem, because in these prior works bandwidth is the only resource to allocate. Hui [101] defined the effective bandwidth of a random variable to be a scaled logarithm of its moment-generating

function. Kleinberg *et al.* [102] provided another definition for some restricted cases and proposed heuristics which give guaranteed approximations for the problem. Our task is then to extend this to the multi-dimensional case.

For a $d$-dimensional random vector $v$, its effective size $\beta_p(v)$ takes as input an allowable overflow probability $p$ and can be either a deterministic $d$-dimensional vector or simply a scalar. Once such a metric is developed, one can turn the problem into a deterministic (multi-dimensional) knapsack problem.

Via comprehensive numerical experiments including settings in which demands follow a mix of different distributions, we show that the general effective bandwidth paired with multi-dimensional knapsack heuristics can generate satisfactory results. However, in many cases, the allocations are conservative. Therefore, we propose an effective way to relax the constraints in the resulting deterministic problem, in order to achieve near-optimal solutions.

Our main contributions are as follows:

1. To our best knowledge, we are the first to look at the fixed-set stochastic multi-dimensional knapsack problem.

2. We provide a simple lower bound and a SAA method-based upper bound for the problem.

3. We extend the effective bandwidth concept to multi-dimensions and use it to solve a knapsack-type chance-constrained program.

4. We conduct numerical experiments to evaluate different combinations of effective bandwidth and knapsack solver.

5. We show that an interesting trade-off of aggressiveness may be performed by relaxing capacity constraints.

The rest of this chapter is organized as follows. Section 6.2 presents related work. Section 6.3 formally defines the problem and provides a lower bound and an upper bound to the optimal solution. Section 6.4 presents algorithms and discuss effective bandwidth for general and special cases. Section 6.5 shows the result of our numerical experiments. Finally, Section 6.6 concludes the chapter.

## 6.2 Related Work

Similar problems have been studied in the literature [103, 104, 13], in which item sizes are also given by some probabilistic distributions. Deal *et al.* [103] show that the optimum profit can be achieved when capacity is slightly violated, while Halman *et al.* [104] show that a $(1+\epsilon)$ approximation can be achieved with a strict knapsack capacity constraint. However, the size of an item is revealed as soon as it is inserted into the knapsack. Then its size is considered fixed. The solution is a particular order of the insertions. The procedure ends when the capacity constraint of the knapsack is violated. In the same setting, the multi-dimensional version and the special case of set packing has also been studied by Dean *et al.* [105]. Our problem is different in that the sizes of items are changing over time even after they are inserted to the knapsack.

The closest work may be referred to as the *fixed set model* of the stochastic knapsack problem. It is motivated by the problem of allocating bandwidth to bursty connections [102], in which the bandwidth is allocated to a maximum profit subset of connections with a bounded *overflow probability.* For some restricted classes of demand distributions, Goel and Indyk [12] gave a PTAS by relaxing the knapsack constraints by a factor of $(1 + \epsilon)$. Recently, Bhalgat *et al.* [13] improved the approximation guarantee for both cases with and without relaxing the knapsack constraints. However, to our best knowledge, the multi-dimensional knapsack problem in the fixed set model has not been studied yet.

Theoretically, solving this problem relates to chance-constrained program introduced by Charnes *et al.* [106] (refer to Prékopa [107] for a comprehensive survey). The problem remained intractable until recently. Promising approximation algorithms include *scenario approximation* [108, 109] and *sample average approximation* (SAA) [99, 100]. These algorithms solve the general problem but are based on Monte-Carlo simulation and involve sampling the problem multiple times. We use the SAA technique only to obtain an upper bound to the optimum in order to evaluate our solutions. Our focus is mainly on developing efficient algorithms for this particular chance-constrained program.

Although good approximation algorithms for the one-dimensional case including PTAS have been proposed in the literature, few simulations and numerical

experiments have been conducted to show how these algorithms perform in practice. Therefore, our goal in this chapter also includes evaluating algorithms via experiments.

## 6.3 Formulation

In this section, we formally define the problem and provide a lower bound and an upper bound to the optimum.

### 6.3.1 Problem Definition

We are given a set $R$ of $d$ resources and a set $T$ of $n$ tasks. Each resource $R_i$ has a size or capacity $b_i$. Each task $T_j$ requests $A_{ij}$ of resource $R_i$ and is associated with a profit $c_j$ if all its demands are satisfied. Let $c = [c_1, c_2..., c_n]$ denote the profit vector of tasks.

$A_{ij}$ is a random vector based on a certain distribution or having some known statistics. We assume that missions are independent in that the requirement of task 1 has nothing to do with that of task 2. That is, for any $i$, $A_{ij}$ and $A_{ik}$ are independent if $j \neq k$. If two tasks have correlated demands, we may consider them as one task and merge their demands.

We use a binary decision variable $x_j$ to indicate whether $T_j$'s demand is satisfied. That is, $x_j = 1$ if $T_j$ is admitted; otherwise, $x_j = 0$. Let $p$ denote the overflow probability, which indicates the maximum frequency that admitted tasks may violate the capacity constraints. The objective is to maximize the profit of admitted tasks with respect to capacity constraints. We formulate the problem as follows:

$$\max \quad \sum_j c_j x_j \tag{6.1}$$

$$\text{subject to} \quad Pr(\sum_j A_{ij} x > b_i) \leq p \qquad \forall i \tag{6.2}$$

This problem is in the form of a *chance-constrained program* [106], and may

be solved by *scenario approximation* [108, 109] or *sample average approximation* (SAA) [99, 100]. It is a hard problem in general; for some distributions such as Bernoulli [102], it is even #P-hard to compute the probability $Pr(\sum_j A_{ij}x > b_i)$.

### 6.3.2 A Simple Lower Bound

Given a distribution of the random demands $A_{ij}$, we could easily calculate the mean value $E[A_{ij}]$. Using the mean value as a size metric and calculating a corresponding deterministic knapsack problem, we can obtain a lower bound to the original problem.

**Theorem 7.** *Solving Equation 6.1 with Equation 6.3 provides a lower bound to the optimum.*

$$\sum_j E[A_{ij}]x_j \leq pb_i \qquad \forall i \tag{6.3}$$

*Proof.* Using Markov Inequality we can get

$$\begin{aligned} Pr(\sum_j A_{ij}x_j > b_i) &\leq Pr(\sum_j A_{ij}x_j \geq b_i) \\ &\leq \frac{E[\sum_j A_{ij}x_j]}{b_i} = \frac{1}{b_i}\sum_j E[A_{ij}]x_j \end{aligned}$$

By enforcing $\frac{1}{b_i}\sum_j E[A_{ij}]x_j \leq p$, we are solving a more restricted problem. □

### 6.3.3 Upper Bound

The upper bound is based on *sample average approximation* with the following idea: we sample the random variable $A_{ij}$ $N$ times and obtain $N$ samples for each $A_{ij}$. Then we generate one problem with all these samples. This problem is larger with $Nd$ additional constraints and decision variables. However, it is a deterministic problem and provides an upper bound to the original problem.

Define $\mathcal{I}(x)$ to be the indicator function:

$$\mathcal{I}(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Let $A_{ij}^1, ..., A_{ij}^N$ be $N$ independent Monte-Carlo samples of the random vector $A_{ij}$. It has been shown in [99, 100] that the following constraints give an upper bound to the original problem for the one-dimensional case:

$$\frac{1}{N}\sum_{k=1}^{N}\mathcal{I}(\sum_j A_j^k x_j - b) \leq \gamma \tag{6.4}$$

where $\gamma > p$. This implies that if we take "slightly more risk" than the overflow probability $p$, the solution will be no worse than the optimal.

We claim that the result carries on to the multi-dimensional case. That is,

**Theorem 8.** *Solving Equation 6.1 with Equation 6.5 provides an upper bound to the original problem.*

$$\frac{1}{N}\sum_{k=1}^{N}\mathcal{I}(\sum_j A_{ij}^k x_j - b_i) \leq \gamma \qquad \forall i \tag{6.5}$$

*Proof.* We omit the proof as it is a direct extension to the one in [99]. □

Using a similar method as [100], we can convert Equation 6.5 to the following Integer Program constraints:

$$\sum_{k=1}^{N} z_i^k \leq N\gamma \qquad \forall i \tag{6.6a}$$

$$\sum_j A_{ij}^k x_j - b_i z_i^k \leq b_i \qquad \forall i, k \tag{6.6b}$$

where $z_i^k \in \{0, 1\}$ is the additional decision variable for sample $k$ in dimension $i$.

To obtain an upper bound with a specified confidence $1 - \delta$, $N$ and $\gamma$ need to

satisfy the following condition as shown in [100]:

$$N \geq \frac{1}{2(\gamma - p)^2} \log \frac{1}{\delta} \tag{6.7}$$

Note that we may even use this method to solve the original problem, as in some cases, the solution to Equation 6.1 and Equation 6.5 leads to a feasible solution to the original problem. However, this is not guaranteed. Besides, $N$ is usually big and we are introducing $N \times d$ new variables and $N \times d$ new constraints to the problem, so the running time of this method may be very long. Therefore, we use it only as an upper bound to evaluate our other algorithms.

## 6.4 Algorithms

In this section, first we propose two frameworks to solve this problem. Then we discuss the effective bandwidth for both general distributions and some special distributions.

The main idea is that we calculate effective bandwidth of the demands vectors, thus convert the problem into a deterministic knapsack problem. Let $\beta_p(X)$ denote the effective bandwidth of random variable/vector $X$ with specified overflow probability $p$. Based on the type of effective bandwidth, the algorithms can be categorized into two frameworks:

1. If the effective bandwidth maps a d-dimensional stochastic vector to a d-dimensional vector, in other words, $\beta_p(X)$ maps each element of a random vector to a scalar, we solve the the following deterministic multi-dimensional knapsack problem:

$$\begin{aligned} \max \quad & \sum_j c_j x_j \\ \text{subject to} \quad & \sum_j \beta_p(A_{ij}) x_j \leq \beta_p(b_i) \qquad \forall i \end{aligned}$$

2. If the effective bandwidth maps a d-dimensional stochastic vector to a single

scalar, we solve the following basic 0/1 knapsack problem.

$$\max \sum_j c_j x_j$$
$$\text{subject to} \quad \sum_j \beta_p(A_j) x_j \leq \beta_p(b)$$

where $A_j = [A_{1j}, A_{2j} ..., A_{dj}]^T$ denote the demand vector of mission $j$ and $b = [b_1, b_2, ..., b_d]^T$.

For the basic 0/1 knapsack problem, we may use the dynamic programming algorithm or a fast approximation heuristic with specified accuracy. The multidimensional knapsack problem, however, is hard to solve in general. For a relatively complete survey on knapsack problem, please refer to the book by Keller *et al.* [110]. For simplicity, we use a slightly modified Adaptive Fixing Heuristic introduced by Bertsimas and Demir [111].

The heuristic solves the LP relaxation first and obtains the optimal solution $x^{LP}$. It fixes $x_j$ to zero if $0 \leq x_j^{LP} < \gamma$, where $\gamma$ is a predetermined threshold (usually set to 0.2); it also fixes $x_j$ to one if $x_j^{LP} = 1$. Then it iteratively solves the LP relaxation with more and more variables fixed until all variables are either zero or one. In case no variables are changed in an iteration and there are some variables satisfying $\gamma < x_k < 1$, we set the minimum $x_k \in (\gamma, 1)$ to zero. Please see Algorithm 16 for details.

Now we discuss several definitions of effective bandwidth.

### 6.4.1 Effective Bandwidth Mapping Random Vectors to Vectors

The concept of effective bandwidth has been widely used in the field of admission control and bandwidth management. Refer to [112] for a relatively complete survey. Applying existing definitions of effective bandwidth to each element of a random vector, we can get a d-dimension to d-dimension mapping. The following three effective bandwidths fall into this category.

**Algorithm 16** Modified Adaptive Fixing Heuristic

1: solve the LP relaxation and obtain $x^{LP}$.
2: **while** $x^{LP}$ includes fractional value **do**
3: **for all** $x_j$ **do**
4: **if** $0 \leq x_j^{LP} < \gamma$ **then**
5: fix $x_j^{LP} \leftarrow 0$.
6: **end if**
7: **if** $x_j^{LP} = 1$ **then**
8: fix $x_j^{LP} \leftarrow 1$.
9: **end if**
10: **end for**
11: **if** no $x_j^{LP}$ is updated **then**
12: $j^* = \underset{0<x_j^{LP}<1}{\text{argmin}} x_j^{LP}$ and fix $x_{j^*}^{LP} = 0$.
13: **end if**
14: solve the LP relaxation with fixed variables.
15: **end while**
16: solution $x \leftarrow x^{LP}$

#### 6.4.1.1 Effective Bandwidth for General Distributions (d-to-d)

For a random variable $X$, its *effective bandwidth* [101, 112] is defined as:

$$\beta_p(X) = \frac{\log E[p^{-X}]}{\log p^{-1}} \tag{6.8}$$

The following proposition has been shown in [101, 102]:

**Proposition 6.4.1.** *Let $X_1, ..., X_n$ be independent random variables, and $X = \sum_j X_j$. Let $b > a$. If $\sum_j \beta_p(X_j) \leq a$, then $Pr[X \geq b] \leq p^{b-a}$.*

In practice, we set $a = b - 1$ and use the following condition:

$$\sum_j \beta_p(A_{ij} x_i) \leq b_i - 1, \tag{6.9}$$

which means $\beta_p(b_i) = b_i - 1$. Then $Pr[\sum_j A_{ij} x_j > b] \leq p$.

This also provides a lower bound to the optimum. We will see in Section 6.5 that this bound is tighter than the mean-based lower bound and is tunable in order to achieve near-optimal solution.

#### 6.4.1.2 Effective Bandwidth for Poisson Trials Only ($\lambda$-based)

Let $\mathcal{P}(\lambda)$ denote the Poisson trial with mean $\lambda$. Poisson variables have the following property:

$$\mathcal{P}(\lambda_1) + \mathcal{P}(\lambda_2) \equiv \mathcal{P}(\lambda_1 + \lambda_2)$$

where $X \equiv Y$ indicates $X$ and $Y$ have identical distributions.

Let $\lambda_{ij}$ denote the mean of $A_{ij}$ if it is a Poisson variable. Let $\mathcal{G}(p, x)$ be the inverse Poisson CDF[1] that outputs a corresponding $\lambda$ given a probability $p$ and a value $x$. The following condition is equivalent to Equation 6.2 for Poisson distribution:

$$\sum_j \lambda_{ij} x_j \leq \mathcal{G}(1-p, b_i) \qquad \forall i \tag{6.10}$$

**Theorem 9.** *Equation 6.10 and Equation 6.1 define the same feasible region for* $x = [x_1, x_2, ..., x_n]$.

*Proof.* By contradiction. Suppose a point $x = [x_1, ..., x_n]$ satisfying Equation 6.10 does not satisfy Equation 6.2, then

$$Pr(\sum_j A_{ij} x_j \leq b_i) > 1 - p$$

However, let $\mathcal{F}(\lambda, x)$ denote the Poisson CDF, which is monotonically decreasing with respect to $\lambda$, then

$$\begin{aligned} Pr(\sum_j A_{ij} x_j \leq b_i) &= \mathcal{F}(\sum_j \lambda_{ij} x_j, b_i) \\ &\geq \mathcal{F}(\mathcal{G}(1-p, b_i), b_i) = 1 - p \end{aligned}$$

Therefore, a point satisfying Equation 6.10 must satisfy Equation 6.2, and vice versa. □

In other words, this "effective bandwidth" transforms the stochastic knapsack problem into a deterministic one with exactly the same feasible region. The disadvantage is, however, that $\mathcal{G}(p, x)$ does not have a closed form, thus we need

[1]This function is an inverse regularized incomplete gamma function.

numerical methods such as Newton's method to calculate $\mathcal{G}(p, x)$. To overcome this, we may precompute a table for frequently used $(1-p, b_i)$ pairs.

#### 6.4.1.3 Effective Bandwidth for Bernoulli Trials Only (Bern)

Kleinberg *et al.* [102] proposed an effective bandwidth for the Bernoulli trial of type $(q, s)$ as the following:

$$\beta'_p(X) = \min\{s, sqp^{-s}\}.$$

where the trial obtains value $s$ with probability $q$ and 0 with probability $1-q$.

They also showed that for a Bernoulli trial, $\beta_p(X) \leq \beta'_p(X)$, which means $\beta'_p(X)$ applies wherever $\beta_p(X)$ applies. However, this also implies that using $\beta'_p(X)$ directly may generate a more constrained problem.

### 6.4.2 Effective Bandwidth Mapping Random Vectors to Scalars (d-to-1)

Basic 0/1 knapsack problems admit simple and efficient solutions. This motivates us in developing an effective bandwidth that maps a random vector to a scalar. The idea is that we use the maximum element of a vector as its delegate, after we apply the general d-dimension to d-dimension mapping. Before we compare the elements, we normalize these demand vectors by the capacity in each dimension.

Formally, for a vector $X = [X_1, X_2, ..., X_d]^T \in R^d$, we define its d-dimensional effective bandwidth $\beta_p^d(X)$ as:

$$\beta_p^d(X) = \max_i[\beta_p(\frac{X_i}{b_i - 1})]$$

and

$$\beta_p^d(b) = 1.$$

The constraints become $\sum_j \beta_p^d(A_j)x_j \leq 1$.

**Theorem 10.** *If $\sum_j \beta_p^d(A_j)x_j \leq 1$, then for any $i$, $Pr[\sum_j A_{ij}x_j > b] \leq p$.*

*Proof.* For any $i$, we have

$$\sum_j \beta_p(A_{ij}x_j)/(b_i - 1) \leq \sum_j \max_i[\beta_p(\frac{A_{ij}}{b_i - 1})]x_j$$
$$= \sum_j \beta_p^d(A_j)x_j \leq 1$$

Therefore, for any $i$, if $\sum_j \beta_p(A_{ij}x_i) \leq b - 1$, then based on Proposition 6.4.1, $Pr[\sum_j A_{ij}x_j > b] \leq p$. □

## 6.5 Numerical Experiments

In this section, we present the results of our numerical experiments.

### 6.5.1 Experiment Setup

Let $U[a, b]$ denote a uniform distribution between $a$ and $b$. In the experiments, there are 10 tasks, each of which has a profit randomly drawn from $U[1, 10]$. We vary the number of resources from 5 to 10 in each series of experiments. For a particular number of resources, we randomly generate 50 problem instances. Therefore, a total of 300 problem instances are generated for each series of experiment.

For a problem instance, each element of the capacity vector $b$ is randomly drawn from $U[2, 10]$. The mean value of every $A_{ij}$ is drawn from $\alpha \times U[1, b_i]$, where $\alpha$ is a tunable parameter that impacts the size of the feasible region.

The solution includes a subset of admitted tasks. To evaluate the obtained profit, we calculate the upper bound using Equation 6.1 and Equation 6.5. All plotted profits in this section are relative profits against the upper bound. Based on Equation 6.7, $N$ increases proportionally to $\frac{1}{2(\gamma - p)^2}$; for a reasonable number of samplings, we set $\gamma = 2p$, $\delta = e^{-2}$ then $N = p^{-2}$ and find it sufficient.

To calculate the overflow probability for a solution, we conduct the following random trials. With a given set of tasks from a solution, we randomly draw 200 samples of their demands and count the number of times when their sum of demands exceeds the capacity limit. The reported overflows are averaged among this 200 samples and over all dimensions.

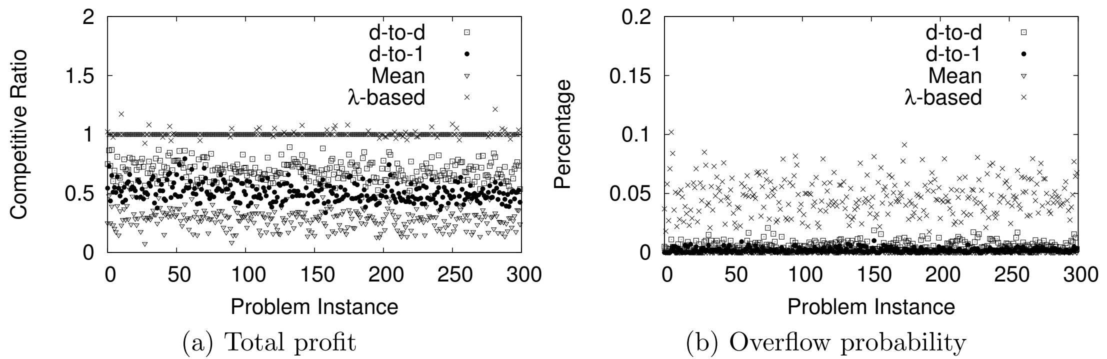
We test the following combinations of effective bandwidth and knapsack solver:

- For mapping d-dimensional random vectors to d-dimensional vectors, we test effective bandwidths for: 1. general distributions (**d-to-d**), 2. Poisson only (**$\lambda$-based**), 3. Bernoulli only (**Bern**) and 4. using mean as the effective bandwidth (**Mean**). Each is paired with Algorithm 16.
- For mapping d-dimensional random vectors to a scalar, we test the effective bandwidth for general distributions (**d-to-1**), paired with a 0/1 knapsack solver.

Note that Mean is based on Theorem 7 and serves as a baseline solution. Now we present the results for each series of tests.

### 6.5.2 Poisson Trials

We plot the result for $p = 15\%$ in Figure 6.1. As we can see in Figure 6.1a, the $\lambda$-based achieved near-optimal solutions as expected. It also implies that the upper bound we use is close to the optimal. There are occasional cases in which the $\lambda$-based is greater than the upper bound; this is because the upper bound has a specified confidence of $1 - \delta = 86.5\%$. The other effective bandwidths in the order of decreasing achieved relative profits are d-to-d, d-to-1 and Mean.

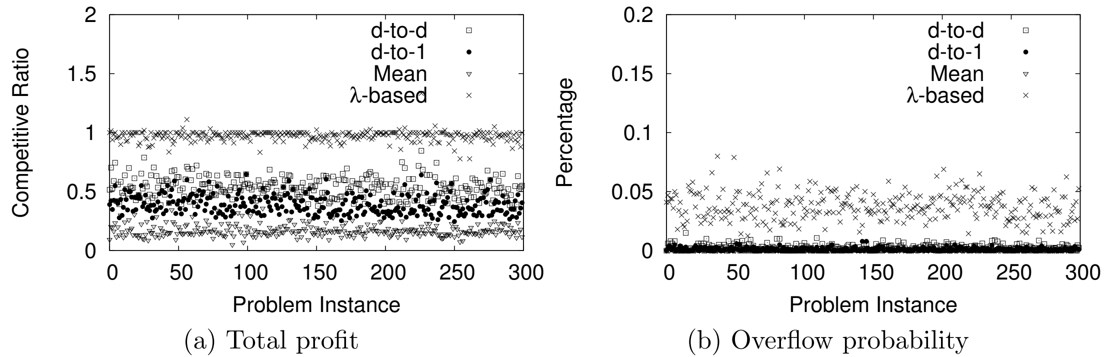



(a) Total profit (b) Overflow probability

Figure 6.1: Poisson, $p = 15\%$

In Figure 6.1b, the $\lambda$-based has an overflow probability ranging from 0 to 10%, less than the limit of 15%. The other three effective bandwidths generate very few overflows and are thus overly conservative. The least conservative is d-to-d.

In the second series of experiments, we set $p = 10\%$. As we can see from Figure 6.2, $\lambda$-based still achieves near-optimal solutions; other effective bandwidths allocate resources more conservatively, among which d-to-d is still the most aggressive.

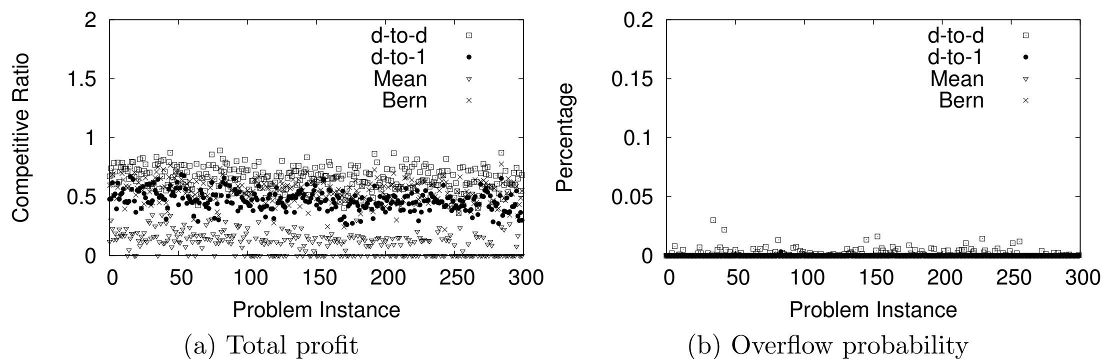



(a) Total profit (b) Overflow probability

Figure 6.2: Poisson, $p = 10\%$

### 6.5.3 Bernoulli Trials

Results for Bernoulli trials with $p = 15\%$ are shown in Figure 6.3. Ranking for relative profit are d-to-d, d-to-1, Bern and Mean. Although the Bernoulli distribution does not have a near-optimal solution like Poisson, the d-to-d effective bandwidth performed on average about 75% of the optimal.




(a) Total profit (b) Overflow probability

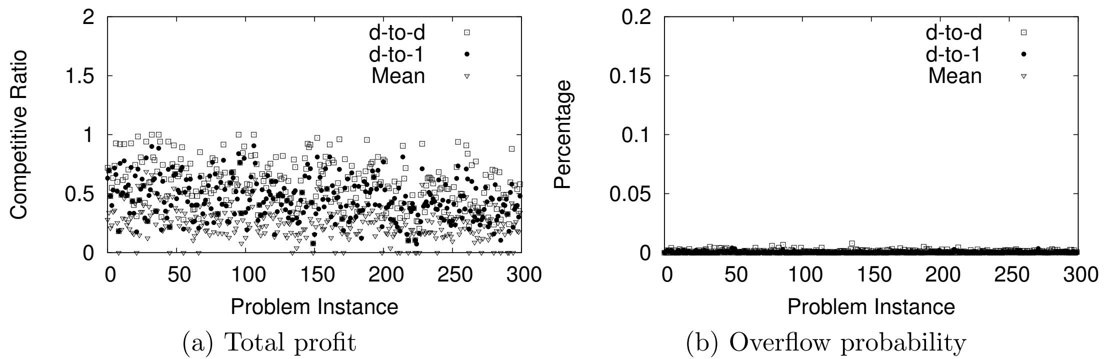
Figure 6.3: Bernoulli, $p = 15\%$

A major difference from the result of Poisson is that the overflow probability is extremely low. Even the most aggressive d-to-d generates only 2% of overflow.

The reason lies in the nature of Bernoulli distribution, as it is an on-off and rather bursty distribution; since the overflow probability constraint is a hard constraint, the allocation tends to be more conservative.

### 6.5.4 Exponential Trials

The allocation for the exponential distribution is also conservative, as we can see from Figure 6.4. The results are similar to those of Bernoulli trials. The relative profit for d-to-d and d-to-1 span from 90% to as bad as 25%. However, the overflow probabilities are very close to zero, which means we may still have space to allocate more tasks.




(a) Total profit (b) Overflow probability

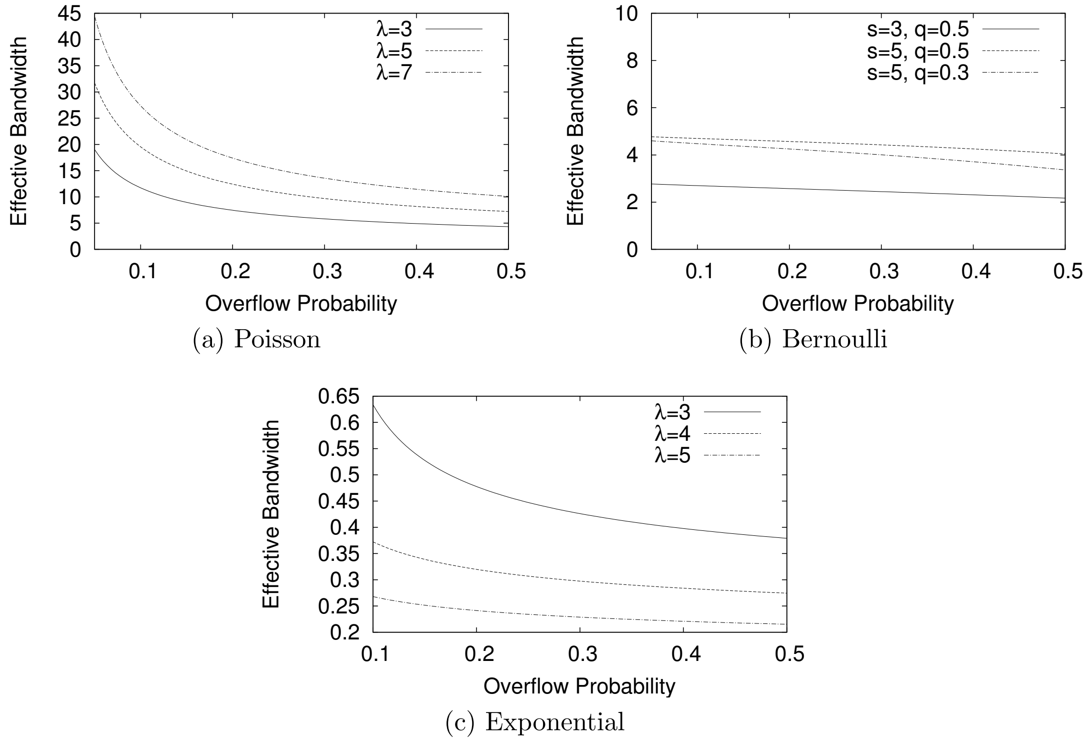
Figure 6.4: Exponential, $p = 15\%$

### 6.5.5 Relaxing the Constraints

We notice that the lower bound by directly applying d-to-d or d-to-1 is not tight. We also notice that the overflow probabilities are very low. This motivates the following experiments.

We attempt to improve the result by relaxing the constraints. The first idea is to increase the overflow probability. However, after we plot $\beta_p(X)$ as a function of $p$, as shown in Figure 6.5, we find that the effective bandwidth only decreases sub-linearly as we increase $p$.

Another way is to relax the capacity constraints $b$ by a constant. Originally, $\beta_p(b) = b - 1$; in the experiments plotted in Figure 6.6, we set $\beta_p(b) = b - 1 + 1.5 = b + 0.5$. This implies that we relax the capacity constraints of the original




(a) Poisson

(b) Bernoulli

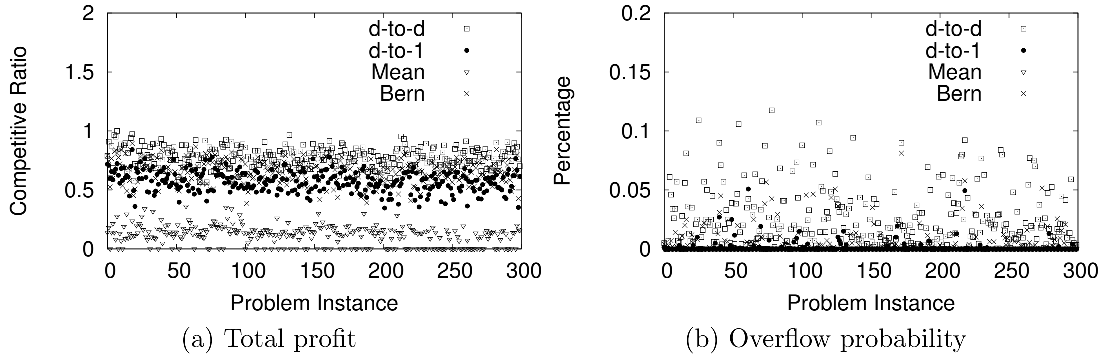
(c) Exponential

Figure 6.5: Impact of $p$ on effective bandwidth $\beta_p(X)$

problem by 1.5 in all dimensions. This may result in an overflow probability that exceeds the theoretical limit, however, as the allocation is generally conservative, it achieved plausible improvement. As we can see in Figure 6.6a, the relative profit, especially for d-to-d, is usually above 60% and at 75% on average. For overflow probability, Figure 6.6b shows that only three instances exceed 15%. We set $\beta_p(b) = b-1+2 = b+1$ for Exponential trials and found a similar improvement.

### 6.5.6 Experiments with a Mix of Different Distributions

General effective bandwidth has the advantage of broad applicability. It has the potential to work with random demands that follow different distributions in different dimensions. Now we present the experiments with demands following a mix of the three distributions. There are 9 resources/dimensions in this experiment. Demands in the first three dimensions follow Poisson distributions, dimension 3 to 6 follow Bernoulli distributions and dimension 7 to 9 are Exponential trials. The

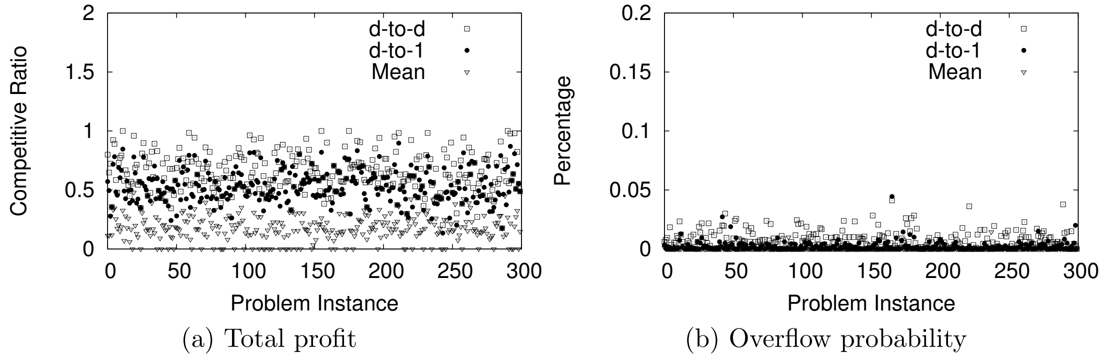



(a) Total profit (b) Overflow probability

Figure 6.6: Bernoulli, $p = 15\%$, $\beta_p(b) = b + 0.5$

rest of the settings are similar to the previous experiments.

We use $\lambda$-based for Poisson and either d-to-d or d-to-1 effective bandwidth for Bernoulli and Exponential. We do not relax any constraints. The results are shown in Figure 6.7. The relative profit in Figure 6.7a shows that d-to-d achieves near-optimal solution and d-to-1's performance is usually above 0.5 of the optimal. Figure 6.7b shows that they produced overflows within the overflow probability 15%.




(a) Total profit (b) Overflow probability

Figure 6.7: A mix of Poisson, Bernoulli and Exponential distributions, $p = 15\%$

To summarize, effective bandwidth-based solutions are generally conservative, especially for bursty distributions. However, we can relax the constraints in order to allocate resources more aggressively. d-to-d effective bandwidth is slightly better than d-to-1, for it is more fine-grained than d-to-1. But d-to-d needs to solve a multi-dimensional knapsack problem. $\lambda$-based for Poisson is expected to be an optimal solution if an optimal solver to the multi-dimensional knapsack problem

is given. The effective bandwidth only for Bernoulli does not perform better than the general d-to-d effective bandwidth, because it is more constrained as discussed in Section 6.3.

## 6.6 Conclusion

In this chapter, we study the problem of allocating multiple resources among a group of users/tasks with stochastic demands. We model this as a stochastic multi-dimensional knapsack problem. We use different combinations of effective bandwidth and knapsack solvers to solve the problem. Via numerical experiments, we found the effective bandwidth for general random trials consistently produces satisfactory results. We also show that by relaxing the constraints of the resulting deterministic knapsack problem, we may achieve near-optimal performance.

We are looking for a tighter bound than Proposition 6.4.1, which may directly lead to near-optimal solutions. Proposition 6.4.1 is derived from the Markov inequality. A Chernoff bound is generally tighter than a Markov bound thus it might be promising to derive a bound with the Chernoff inequality.

Chapter 7

# Conclusions

In this chapter, we first summarize the research approach applied in this dissertation. Then we highlight some contributions in this dissertation. Finally, we propose some future directions.

## 7.1 General Approach

The topics addressed in this dissertation span a wide range of networks, but the general approach in solving these problems are similar: modeling, analysis, algorithm design and evaluation.

We first develop a mathematical model of a real world problem with reasonable assumptions. Linear or Integer Programs are often used, because they not only help with analysis and give insight into the solution, but also provide either the optimal or a bound to the optimal for the purpose of evaluation.

Challenges then lie in the complex nature of the current networked systems, such as modeling the heterogeneity of a large scale network or modeling interference and mobility of a wireless network. Although simplification is inevitable in modeling complex systems, by carefully making assumptions we may obtain a tractable abstract problem without losing adequate factors that lead to practicable solutions.

Optimization aims to develop computational methodologies for maximizing an objective subject to constraints. For networks, typical objectives such as the *utilization* of resources are usually in the form of sub-modular or even linear functions,

which may lead to simple and efficient solutions. It is the complexity of constraints of a real system that make the optimization difficult. Fortunately, a large number of these abstract problems (or similar problems) have solutions in the literature. Although the focus of this dissertation is not purely on theory, algorithm design that extends theoretical results and proof of performance guarantees have often contributed to the depth of this work.

Sound theoretical performance needs to be verified in realistic settings. The difficulty of verification, either through implementation or simulation, lies in the creation of a realistic networking environment. This is done by carefully designing the evaluation logic, controlling variable factors and comparing with existing experimented data.

## 7.2 Contributions

This dissertation attempted to solve a class of problems of allocating network resources in several information-centric scenarios. These problems seek to maximize the utilization of network resources and provide guaranteed performance using assignment and scheduling methods with respect to realistic but complex constraints. These problems are in general difficult to solve (NP-hard in most cases). Therefore, our goal is to provide efficient solutions to approximate the optimal solution. Also for the purpose of scaling, we often design and adapt our solutions to work in a distributed way. The solutions, evaluated via simulation or trace-based study, usually performed well as expected.

In Chapter 2 we considered a specialized content distribution application in wireless mesh networks [113, 114]. When a new *mission* arrives, data is pushed to storage nodes at the mission site where it may be retrieved locally. We developed efficient heuristics to choose a storage node assignment minimizing the total latency-based cost. In NS2 simulations, we find that our heuristic algorithms achieve on average a cost within at most 15% of the optimum. We also defined distributed protocols to implement these algorithms and show that they exhibit acceptable overheads even as the problem size grows.

In Chapter 3 we considered variations of a problem in which data must be delivered to mobile clients en-route, as they travel towards their destinations [115,

116, 117]. We cast this scenario as a parallel-machine scheduling problem with the little-studied property that jobs may have different release times and deadlines when assigned to different machines. We presented new algorithms and also adapt existing algorithms, for both online and offline settings. We evaluated these algorithms on a variety of problem instance types, using both synthetic and real-world data, including several geographical scenarios, and show that our algorithms produce schedules achieving near-optimal throughput.

In Chapter 4 we investigated the joint problem of building distribution paths and placing Web server replicas in cloud CDNs to minimize the cost incurred on the CDN providers while satisfying QoS requirements for user requests [118]. We showed that the monthly cost can be as low as 2.62 US Dollars for a small Web site. We developed a suite of offline, online-static and online-dynamic heuristic algorithms and evaluated them via Web trace-based simulation. We showed that our heuristics behave very close to optimal under various network conditions.

In Chapter 5 we extended the Convergecast problem in two ways [119]. First, we studied a) one-shot throughput maximization in settings with aggregation and b) periodic scheduling in settings without aggregation. Second, we generalized the notion of aggregatability in both one-shot and periodic scheduling beyond the binary choice of either all sets of items being aggregatable or none being so. We provided optimal algorithms, guaranteed approximations and heuristics for a variety of general and special cases. We then evaluated the algorithms in a systematic simulation study, both under the conditions in which our provable guarantees apply and in more general settings, where we find the algorithms continue to perform well on typical problem inputs.

In Chapter 6, we studied the problem of allocating multiple resources among a group of users/tasks with stochastic demands [120]. The goal is to admit as many users as possible to the system without violating the resource capacity more often than a predefined overflow probability. We modeled this as a stochastic multi-dimensional knapsack problem. We provided bounds on the optimal solution. Then we extended and applied the concept of effective bandwidth in order to solve this problem efficiently. Via numerical experiments, we show that our algorithms achieve near-optimal performance with specified overflow probability.

## 7.3 Future Directions

Information-centric networking is a novel and developing networking paradigm in which information objects are the first-class abstraction and the unit for networking. This paradigm requires the network to support functionalities such as in-network storage, caching, multicasting, publish-subscribe and QoS. It also requires the network to be highly scalable, easy to configure, secure and low cost. These requirements may lead to many challenging research problems.

One direction to pursue is to capture the system dynamics by scheduling. The incompatibility of users in terms of resource requirements may be expressed by a conflict graph. A user requires different resources at different stages. Two users are compatible if their conflicting demands are scheduled to be satisfied at different times. Therefore, a scheduling of the usage over time may further eliminate the chance of overflow in demands and thus can admit more users than the fixed assignment. The problem can be further extended to take into account the precedence constraints within one user's own demands.

Another direction is to study the habits of users, such as data preferences and mobility patterns. To do scheduling and assignment, this dissertation usually presumes the knowledge of such information. In practice, however, obtaining this information requires techniques such as data mining and machine learning. A study on learning users' habit and predicting their future needs and movements may lead to more efficient and accurate data delivery and dissemination.

# Bibliography


[1] Koponen, T., M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica (2007) "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM Computer Communication Review*, pp. 181–192.

[2] Krishnamurthy, B., C. Wills, and Y. Zhang (2001) "On the use and performance of content distribution networks," *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 169–182.

[3] Intanagonwiwat, C., R. Govindan, D. Estrin, J. Heidemann, and F. Silva (2003) "Directed diffusion for wireless sensor networking," *Networking, IEEE/ACM Transactions on*, **11**(1), pp. 2–16.

[4] Deering, S. and D. Cheriton (1990) "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, **8**(2), pp. 85–110.

[5] Qiu, L., V. Padmanabhan, and G. Voelker (2001) "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM 2001*.

[6] Rodolakis, G., S. Siachalou, and L. Georgiadis (2006) "Replicated server placement with QoS constraints," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1151–1162.

[7] Tang, X. and J. Xu (2005) "QoS-aware replica placement for content distribution," *IEEE Transactions on Parallel and Distributed Systems*, **16**(10), pp. 921–932.

[8] Wang, H., P. Liu, and J. Wu (2006) "A QoS-aware heuristic algorithm for replica placement," in *7th IEEE/ACM International Conference on Grid Computing*.

[9] Ratnasamy, S., B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu (2003) "Data-centric storage in sensornets with GHT, a geographic hash table," *Mobile networks and applications*, **8**(4), pp. 427–442.

[10] Ross, G. T. and R. M. Soland (1975) "A branch and bound algorithm for the generalized assignment problem," *Math. Program.*, **8-1**.

[11] Brucker, P. (2007) *Scheduling Algorithms*, 5th ed., Springer.

[12] Goel, A. and P. Indyk (1999) "Stochastic load balancing and related problems," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, IEEE, pp. 579–586.

[13] Bhalgat, A., A. Goel, and S. Khanna (2011) "Improved approximation results for stochastic knapsack problems," in *SODA*.

[14] Hosseini, M., D. Ahmed, S. Shirmohammadi, and N. Georganas (2007) "A survey of application-layer multicast protocols," *IEEE Communications Surveys & Tutorials*, **9**(3), pp. 58–74.

[15] Baev, I., R. Rajaraman, and C. Swamy (2008) "Approximation Algorithms for Data Placement Problems," *SIAM J. Comput.*, **38(4)**, pp. 1411–1429.

[16] Alzoubi, H. A., S. Lee, M. Rabinovich, O. Spatscheck, and J. E. van der Merwe (2008) "Anycast CDNS revisited," in *Proceedings of WWW 2008*.

[17] Goel, A., K. Ramakrishnan, D. Kataria, and D. Logothetis (2001) "Efficient computation of delay-sensitive routes from one source to all destinations," in *INFOCOM*.

[18] Goel, A. and K. Munagala (2002) "Extending greedy multicast routing to delay sensitive applications," *Algorithmica*, **33**(3), pp. 335–352.

[19] Cohen, R., L. Katzir, and D. Raz (2006) "An efficient approximation for the Generalized Assignment Problem," *Inf. Process. Lett.*, **100(4)**, pp. 162–166.

[20] Lehmann, B., D. J. Lehmann, and N. Nisan (2001) "Combinatorial auctions with decreasing marginal utilities," in *ACM Conference on Electronic Commerce*, pp. 18–28.

[21] Fleischer, L., M. X. Goemans, V. S. Mirrokni, and M. Sviridenko (2006) "Tight approximation algorithms for maximum general assignment problems," in *Proceedings of SODA 06*, pp. 611–620.

[22] Meyerson, A., K. Munagala, and S. A. Plotkin (2001) "Web caching using access statistics," in *SODA*, pp. 354–363.

[23] Guha, S. and K. Munagala (2002) "Improved algorithms for the data placement problem," in *SODA*, pp. 106–107.

[24] Shmoys, D. B. and va Tardos (1993) "An approximation algorithm for the generalized assignment problem," *Math. Program.*, **62**.

[25] Lenstra, J. K., D. B. Shmoys, and va Tardos (1990) "Approximation Algorithms for Scheduling Unrelated Parallel Machines," *Math. Program.*, **46**.

[26] Bateni, M. H. and M. T. Hajiaghayi (2009) "Assignment Problem in Content Distribution Networks: Unsplittable Hard-Capacitated Facility Location," in *Proceedings of SODA 09*.

[27] Bose, P., P. Morin, I. Stojmenović, and J. Urrutia (2001) "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks," *Wireless Networks*, **7**(6), pp. 609–616.

[28] Karp, B. and H. T. Kung (2000) "GPSR: greedy perimeter stateless routing for wireless networks," *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 243–254.

[29] De, S., A. Caruso, T. Chaira, and S. Chessa (2006) "Bounds on hop distance in greedy routing approach in wireless ad hoc networks," *International Journal of Wireless and Mobile Computing*, **1**(2), pp. 131–140.

[30] Jing, J., A. S. Helal, and A. Elmagarmid (1999) "Client-server computing in mobile environments," *ACM Computing Surveys*, **31**(2).

[31] Ott, J. and D. Kutscher (2004) "Drive-thru Internet: IEEE 802.11 b for," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, IEEE.

[32] Bychkovsky, V., B. Hull, A. Miu, H. Balakrishnan, and S. Madden (2006) "A measurement study of vehicular internet access using in situ Wi-Fi networks," in *Proceedings of the 12th annual international conference on Mobile computing and networking*, ACM, p. 61.

[33] Zhang, Y., J. Zhao, and G. Cao (2010) "Service Scheduling of Vehicle-Roadside Data Access," *Mobile Networks and Applications*, **15**(1), pp. 83–96.

[34] Burgess, J., B. N. Levine, R. Mahajan, J. Zahorjan, A. Balasubramanian, A. Venkataramani, Y. Zhou, B. Croft, N. Banerjee, M. Corner, and D. Towsley (2008), "CRAWDAD data set umass/diesel (v. 2008-09-14)," Downloaded from http://crawdad.cs.dartmouth.edu/umass/diesel.

[35] Lee, U., E. Magistretti, M. Gerla, P. Bellavista, and A. Corradi (2009) "Dissemination and Harvesting of Urban Data Using Vehicular Sensing Platforms," *IEEE transactions on vehicular technology*, **58**(2), pp. 882–901.

[36] Balasubramanian, A., B. Levine, and A. Venkataramani (2008) "Enhancing interactive web applications in hybrid networks," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, ACM, pp. 70–80.

[37] Jiang, S. and N. Vaidya (1999) "Scheduling data broadcast to impatient users," in *Proceedings of the 1st ACM international workshop on Data engineering for wireless and mobile access*, ACM, p. 59.

[38] Xu, J., X. Tang, and W. Lee (2006) "Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation," *IEEE Transactions on Parallel and Distributed Systems*, pp. 3–14.

[39] Lee, Y. and H. Sherali (1994) "Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battle-group problem," *Annals of Operations Research*, **50**.

[40] Simons, B. and M. Warmuth (1989) "A Fast Algorithm for Multiprocessor Scheduling of Unit-Length Jobs," *SIAM Journal of Computing*, **18**(4).

[41] He, R. (2005) "Parallel machine scheduling problem with time windows: A constraint programming and tabu search hybrid approach," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*.

[42] Bar-Noy, A., S. Guha, J. Naor, and B. Schieber (2001) "Approximating the Throughput of Multiple Machines in Real-Time Scheduling," *SIAM Journal of Computing*, **31**(2).

[43] Berman, P. and B. DasGupta (2000) "Multi-phase Algorithms for Throughput Maximization for Real-Time Scheduling," *Journal of Combinatorial Optimization*, **4**(3).

[44] Hall, N. G. and M. J. Magazine (1994) "Maximizing the value of a space mission," *European journal of operational research*, **78**(2), pp. 224–241.

[45] Dyer, M. E. and L. A. Wolsey (1990) "Formulating the single machine sequencing problem with release dates as a mixed integer program," *Discrete Applied Mathematics*, **26**(2-3), pp. 255–270.

[46] "A Modeling Language for Mathematical Programming," .
URL http://www.ampl.com/

[47] "IBM ILOG CPLEX Optimizer ," .
URL http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

[48] van den Akker, J. M., C. A. J. Hurkens, and M. W. P. Savelsbergh (2000) "Time-Indexed Formulations for Machine Scheduling Problems: Column Generation," *INFORMS Journal on Computing*, **12**(2), pp. 111–124.

[49] Brucker, P. and S. A. Kravchenko (1999) "Preemption can make parallel machine scheduling problems hard," *Osnabrucker Schriften zur Mathematik, Reihe P.*

[50] Rothkopf, M. H. (1966) "Scheduling Independent Tasks on Parallel Processors," *Management Science*, **12**(5), pp. 437–447.

[51] Lawler, E. L. and J. M. Moore (1969) "A functional equation and its application to resource allocation and sequencing problems," *Management Science*, **16**(1), pp. 77–84.

[52] Broch, J., D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva (1998) "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, ACM New York, NY, USA, pp. 85–97.

[53] Armbrust, M., A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. (2009) *Above the clouds: A berkeley view of cloud computing*, *Tech. rep.*, EECS Department, University of California, Berkeley.

[54] Broberg, J., R. Buyya, and Z. Tari (2009) "MetaCDN: Harnessing Storage Clouds' for high performance content delivery," *Journal of Network and Computer Applications*, **32**(5), pp. 1012–1022.

[55] Broberg, J., S. Venugopal, and R. Buyya (2008) "Market-oriented grids and utility computing: The state-of-the-art and future directions," *Journal of Grid Computing*, **6**(3), pp. 255–276.

[56] Rabinovich, M. and O. Spatschek (2002) *Web caching and replication*, Addison-Wesley Longman Publishing Co., Inc.

[57] Li, B., M. Golin, G. Italiano, and X. Deng (1999) "On the optimal placement of web servers in the Internet," in *Proc. IEEE Infocom*.

[58] Krishnan, P., D. Raz, and Y. Shavitt (2000) "The cache location problem," *IEEE/ACM Transactions on Networking (TON)*, **8**(5).

[59] Radoslavov, P., R. Govindan, and D. Estrin (2002) "Topology-informed Internet replica placement," *Computer Communications*, **25**(4), pp. 384–392.

[60] Jamin, S., C. Jin, A. Kurc, D. Raz, and Y. Shavitt (2001) "Constrained mirror placement on the Internet," in *IEEE INFOCOM*, vol. 1, pp. 31–40.

[61] Xu, J., B. Li, and D. Lee (2002) "Placement problems for transparent data replication proxy services," *IEEE JSAC*, **20**(7), pp. 1383–1398.

[62] Jia, X., D. Li, X. Hu, W. Wu, and D. Du (2003) "Placement of web-server proxies with consideration of read and update operations on the internet," *The Computer Journal*, **46**(4), p. 378.

[63] Cidon, I., S. Kutten, and R. Soffer (2002) "Optimal allocation of electronic content," *Computer Networks*, **40**(2), pp. 205–218.

[64] Kalpakis, K., K. Dasgupta, and O. Wolfson (2001) "Optimal placement of replicas in trees with read, write, and storage costs," *IEEE Transactions on Parallel and Distributed Systems*, pp. 628–637.

[65] Bartolini, N., F. Lo, and P. Petrioli (2003) "Optimal dynamic replica placement in content delivery networks," in *Proc. IEEE Int. Conf. on Networking*, pp. 125–130.

[66] Lo Presti, F., C. Petrioli, and C. Vicari (2005) "Dynamic replica placement in content delivery networks," in *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 351–360.

[67] Vicari, C., C. Petrioli, and F. Presti (2007) "Dynamic replica placement and traffic redirection in Content Delivery Networks," *ACM SIGMETRICS Performance Evaluation Review*, **35**(3), pp. 66–68.

[68] Loukopoulos, T., N. Tziritas, P. Lampsas, and S. Lalis (2007) "Implementing Replica Placements: Feasibility and Cost Minimization," in *IEEE International Parallel and Distributed Processing Symposium*, p. 126.

[69] Forestiero, A., C. Mastroianni, and G. Spezzano (2008) "QoS-based dissemination of content in Grids," *Future Generation Computer Systems*, **24**(3), pp. 235–244.

[70] Lv, Q., P. Cao, E. Cohen, K. Li, and S. Shenker (2002) "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th International conference on Supercomputing*.

[71] Lund, C. and M. Yannakakis (1994) "On the hardness of approximating minimization problems," *Journal of the ACM (JACM)*, **41**(5), pp. 960–981.

[72] Chvatal, V. (1979) "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, pp. 233–235.

[73] Madhyastha, H., T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani (2006) "iPlane: An information plane for distributed services," in *Proceedings of the 7th symposium on Operating systems design and implementation*, USENIX Association, pp. 367–380.

[74] "GeoLite City database by MaxMind," .
URL `http://www.maxmind.com/`

[75] Shavitt, Y. and N. Zilberman (2010) "A Structural Approach for PoP Geo-Location," in *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, IEEE, pp. 1–6.

[76] Padmanabhan, V. and L. Subramanian (2001) "An investigation of geographic mapping techniques for internet hosts," *ACM SIGCOMM Computer Communication Review*, **31**(4), p. 185.

[77] Chintalapudi, K. and L. Venkatraman (2008) "On the design of mac protocols for low-latency hard real-time discrete control applications over 802.15. 4 hardware," in *Proceedings of ACM/IEEE IPSN*.

[78] Ghosh, A., O. Incel, V. Kumar, and B. Krishnamachari (2009) "Multi-channel scheduling algorithms for fast aggregated convergecast in sensor networks," in *Proceedings of IEEE MASS*.

[79] Korteweg, P. (2006) *Online gathering algorithms for wireless networks*, Ph.D. thesis, Eindhoven University of Technology.

[80] Li, H., P. J. Shenoy, and K. Ramamritham (2005) "Scheduling Messages with Deadlines in Multi-Hop Real-Time Sensor Networks," in *Proceedings of IEEE RTAS*.

[81] Hajek, B. and G. Sasaki (1988) "Link scheduling in polynomial time," *IEEE Transactions on Information Theory*, **34**, pp. 910–917.

[82] Ramanathan, S. (1999) "A unified framework and algorithm for channel assignment in wireless networks," *Wireless Networks*, **5**(2), pp. 81–94.

[83] Gandham, S., Y. Zhang, and Q. Huang (2008) "Distributed time-optimal scheduling for convergecast in wireless sensor networks," *Computer Networks*, **52**, pp. 610–629.

[84] Ergen, S. and P. Varaiya (2009) "TDMA scheduling algorithms for wireless sensor networks," *Wireless Networks*, pp. 1–13.

[85] Choi, H., J. Wang, and E. A. Hughes (2008) "Scheduling for information gathering on sensor network," *Wireless Networks*, **15**(1), pp. 127–140.

[86] Gargano, L. and A. Rescigno (2006) "Optimally fast data gathering in sensor networks," *Mathematical Foundations of Computer Science 2006*, pp. 399–411.

[87] Yu, B., J. Li, and Y. Li (2009) "Distributed Data Aggregation Scheduling in Wireless Sensor Networks," in *Proceedings of IEEE INFOCOM*.

[88] Xu, X., S. Wang, X. Mao, S. Tang, and X. Li (2010) "A delay efficient algorithm for data aggregation in multi-hop wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*.

[89] Intanagonwiwat, C., R. Govindan, and D. Estrin (2000) "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of ACM/IEEE MOBICOM*.

[90] Moscibroda, T. (2007) "The worst-case capacity of wireless sensor networks," in *Proceedings of ACM/IEEE IPSN*.

[91] HART Communications Foundation (2007), "Hart protocol specifications, revision 1.0," .
URL `http://www.hartcomm.org`

[92] Zuckerman, D. (2007) "Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number," *Theory of Computing*, **3**(1), pp. 103–128.

[93] Luby, M. (1986) "A Simple Parallel Algorithm for the Maximal Independent Set Problem," *SIAM J. Comput.*, **15**(4), pp. 1036–1053.

[94] Schneider, J. and R. Wattenhofer (2008) "A log-star distributed maximal independent set algorithm for growth-bounded graphs," in *Proceedings of ACM PODC*.

[95] Clark, B. N., C. J. Colbourn, and D. S. Johnson (1990) "Unit disk graphs," *Discrete Mathematics*, **86**(1-3), pp. 165–177.

[96] Hochbaum, D. (1983) "Efficient bounds for the stable set, vertex cover, and set packing problems," *Mathematics*, **6**, pp. 243–254.

[97] Li, X. and Y. Wang (2002) "Simple heuristics and PTASs for intersection graphs in wireless ad hoc networks," in *Proceedings of ACM DIAL-M*.

[98] Kelly, F., S. Zachary, and I. Zeidins (1996) *Stochastic Networks: Theory and Applications*, Oxford University Press.

[99] Luedtke, J. and S. Ahmed (2008) "A sample approximation approach for optimization with probabilistic constraints," *SIAM Journal on Optimization*, **19**(2), pp. 674–699.

[100] Pagnoncelli, B., S. Ahmed, and A. Shapiro (2009) "Sample average approximation method for chance constrained programming: theory and applications," *Journal of optimization theory and applications*, **142**(2), pp. 399–416.

[101] Hui, J. (1988) "Resource allocation for broadband networks," *Selected Areas in Communications, IEEE Journal on*, **6**(9), pp. 1598–1608.

[102] Kleinberg, J., Y. Rabani, and É. Tardos (1997) "Allocating bandwidth for bursty connections," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ACM, pp. 664–673.

[103] Dean, B., M. Goemans, and J. Vondrák (2008) "Approximating the stochastic knapsack problem: The benefit of adaptivity," *Mathematics of Operations Research*, **33**(4), pp. 945–964.

[104] Halman, N., D. Klabjan, C. Li, J. Orlin, and D. Simchi-Levi (2008) "Fully polynomial time approximation schemes for stochastic dynamic programs," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, pp. 700–709.

[105] Dean, B., M. Goemans, and J. Vondrák (2005) "Adaptivity and approximation for stochastic packing problems," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, pp. 395–404.

[106] Charnes, A., W. Cooper, and G. Symonds (1958) "Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil," *Management Science*, **4**(3), pp. 235–263.

[107] Prékopa, A. (1995) *Stochastic programming*, vol. 324, Springer.

[108] Calafiore, G. and M. Campi (2006) "The scenario approach to robust control design," *Automatic Control, IEEE Transactions on*, **51**(5), pp. 742–753.

[109] Nemirovski, A. and A. Shapiro (2006) "Scenario approximations of chance constraints," *Probabilistic and randomized methods for design under uncertainty*, pp. 3–47.

[110] Kellerer, H., U. Pferschy, and D. Pisinger (2004) *Knapsack problems*, Springer Verlag.

[111] Bertsimas, D. and R. Demir (2002) "An approximate dynamic programming approach to multidimensional knapsack problems," *Management Science*, pp. 550–565.

[112] Kelly, F. (1996) "Notes on effective bandwidths," *Stochastic networks: theory and applications*, **4**, pp. 141–168.

[113] Chen, F., M. P. Johnson, A. Bar-Noy, I. Fermin, and T. F. La Porta (2009) "Proactive Data Dissemination to Mission Sites," in *IEEE SECON*.

[114] Chen, F., M. Johnson, A. Bar-Noy, and T. La Porta (2012) "Proactive data dissemination to mission sites," *Wireless Networks*, pp. 1–14.

[115] Chen, F., M. P. Johnson, Y. Alayev, A. Bar-Noy, and T. F. La Porta (2009) "Who, When, Where: Timeslot Assignment to Mobile Clients," in *IEEE MASS*.

[116] Chen, F., M. P. Johnson, A. Bar-Noy, and T. F. La Porta (2010) "Cooperative Data Dissemination to Mission Sites," in *Proceedings of SPIE*.

[117] Chen, F., M. Johnson, Y. Alayev, A. Bar-Noy, and T. La Porta (2012) "Who, When, Where: Timeslot Assignment to Mobile Clients," *IEEE Transactions on Mobile Computing*, **11**(1), pp. 73 –85.

[118] Chen, F., K. Guo, J. Lin, and T. F. La Porta (2012) "Intra-cloud Lightning: Building CDNs in the Cloud," in *IEEE INFOCOM*.

[119] Chen, F., M. P. Johnson, A. Bar-Noy, and T. F. La Porta (2012) "Convergecast with Aggregatable Data Classes," in *IEEE SECON*.

[120] Chen, F., T. F. La Porta, and M. B. Srivastava (2012) "Resource Allocation with Stochastic Demands," in *IEEE DCOSS*.

# Vita

**Fangfei Chen**

## Education

**The Pennsylvania State University**, University Park, PA. Ph.D. in Computer Science and Engineering, Summer 2012.
**Tsinghua University**, Beijing, China. B.S. in Electrical Engineering, Fall 2006.

## Experience

**Research Assistant**, *The Pennsylvania State University*, University Park, PA. 2006–2012.

- Studied resource allocation problems in networking systems.

**Intern**, *Bell Laboratories, Alcatel-Lucent*, Holmdel, NJ. Summer 2011.

- Studied job-scheduling problem in MapReduce systems by explicitly considering job-precedence in Map, Shuffle and Reduce phases.

**Intern**, *Bell Laboratories, Alcatel-Lucent*, Murray Hill, NJ. Summer 2010.

- Solved replica placement problem in building a Content Distribution Network in the cloud.

**Teaching Assistant**, *The Pennsylvania State University*, University Park, PA. 2006.

- Assisted CSE 431 Computer Architecture, graded assignments and instructed lab experiments.

**Intern**, *Synway Information Engineering Co.*, Hangzhou, China. Summer 2005.

- Designed circuits and programmed FPGA for a PCI board that transfers data between DSP chips and a PC using DMA.

## Honors and Awards

- IEEE INFOCOM Student Travel Grant, 2012
- AT&T Graduate Fellowship, 2011
- IEEE MASS Student Travel Grant, 2009
- Tsinghua Second-class Scholarship, 2003