

2005

Data broadcast scheduling: Models, algorithms, and analysis

Aaron Thomas Hawkins
College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Hawkins, Aaron Thomas, "Data broadcast scheduling: Models, algorithms, and analysis" (2005).
Dissertations, Theses, and Masters Projects. Paper 1539623465.
<https://dx.doi.org/doi:10.21220/s2-0er6-b181>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

DATA BROADCAST SCHEDULING: MODELS, ALGORITHMS,
AND ANALYSIS

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

by

Aaron Thomas Hawkins

2005

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy



Aaron T. Hawkins

Approved, April 2005



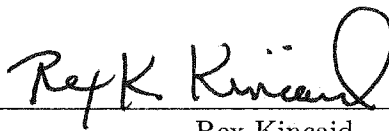
Weizhen Mao
Dissertation Advisor



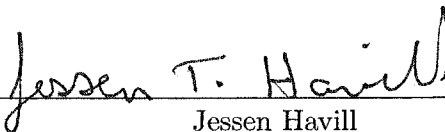
Nikos Chrisochoides



Robert Noonan



Rex Kincaid



Jessen Havill
Denison University

To my parents for always being there, to my daughter for crying when I go to work and smiling when I come home, and to my wife for helping me build a better life than any man deserves.

Table of Contents

Acknowledgments	vii
List of Tables	viii
List of Figures	ix
Abstract	xi
1 Introduction	2
1.1 Motivation	2
1.2 General Problem Description	5
1.3 Formal Problem Definition	9
1.3.1 The Server's System Environment	11
1.3.2 Request Characteristics	13
1.3.3 The Objective Function	14
1.4 The Data Broadcasting Schedule Diagram	15
1.5 Thesis Organization and Contributions	18
2 Related Work	19

2.1	On-Line Algorithms and Competitive Analysis	19
2.2	Data Broadcasting System Environments	23
3	Minimizing Total Wait Time: The Multi-Channel Case	29
3.1	Model Description	30
3.2	Algorithms	30
3.3	Analytical Results	33
3.3.1	First Come First Served (FCFS)	34
3.3.2	Most Requested First (MRF)	37
3.3.3	General Lower Bound for c Channels	39
3.4	Simulation Results	43
3.4.1	Simulation Parameters	44
3.4.2	Performance Comparisons	45
3.4.3	Simulation Conclusions	50
4	Minimizing Two Metrics: The Single-Item Case	53
4.1	Model Description	54
4.2	Algorithms	55
4.3	Analytical Results	62
4.3.1	Competitive Ratio for the LAZY Algorithm	63
4.3.2	Competitive Ratio for the GREEDY Algorithm	69
4.4	Simulation Results	71
4.4.1	Simulation Parameters	72
4.4.2	Performance Comparisons	73

4.4.3	Simulation Conclusions	75
5	Minimizing Two Metrics: The Multi-Item Case	77
5.1	Model Description	77
5.2	Algorithms	79
5.3	Analytical Results	82
5.3.1	A Lower Bound for GREEDY	82
5.4	Simulation Results	84
5.4.1	Simulation Parameters	85
5.4.2	Performance Comparisons	86
5.4.3	Simulation Conclusions	88
6	Conclusions and Future Work	90
	Bibliography	95
	Vita	99

ACKNOWLEDGMENTS

The work contained in this thesis was supported by funding from the Dean's Fellowship, College of William and Mary; the Department of Computer Science Teaching Fellowship, College of William and Mary; and The Virginia Space Grant, Virginia Space Grant Consortium.

The simulations found herein were performed using the computational resources within the Department of Computer Science, College of William and Mary. These resources were established by grants from Sun Microsystems, the National Science Foundation, and the state of Virginia's Commonwealth Technology Research Fund. Special honor is given to the late Steve Park whose instruction, advise, and materials were invaluable in the construction of these simulations long after his passing.

The author wishes to thank Ben Coleman, Leonidas Linardakis, and Paul Stockmeyer for serving as a theoretical support group. The mad diagrams on white boards, equations on napkins, and casual office conversations helped keep the creative juices flowing without ever encountering the phrase "Why would you want to know?".

Grateful acknowledgment is given to Marty Gilbert and Erik Nylander for technical and mathematical support on-call away 24 hours a day. They helped alleviate the difficulties of writing a thesis several hundred miles away from college resources.

Sincere appreciation is extended to Vanessa Godwin for making sure all the "petty details" (such as showing up to teach class, registering for enrollment, and successfully graduating) were addressed throughout the author's tenure as a graduate student. The number of timely e-mails vigilantly sent out from her account is highly correlated with the number of deadlines that were met.

And much thanks goes to colleagues at Rockwell Scientific for being both the carrot and the stick encouraging graduation. They have shown that getting one's Ph.D. is a good place to start your learning career.

Finally, the author wishes to thank his committee members Nikos Chrisochoides, Robert Noonan, Rex Kincaid, and Jessen Havill for bending personal schedules in order to provide quality feedback on the ones found herein. Their willingness to correspond remotely in response to e-mails sent *ad nauseam*, to loan materials for indefinite periods of time, and to even meet during lunch breaks when necessary made this process as painless as possible. The utmost gratitude is especially given to the author's research advisor, Weizhen Mao, whose guidance never wavered in patience, sincerity, or insight. Thanks for always asking how I was doing before asking what I had done for the week.

List of Tables

4.1	EXAMPLE CONSTRUCTION of C1	59
4.2	EXAMPLE CONSTRUCTION OF C	60
4.3	EXAMPLE CONSTRUCTION OF T	61
4.4	INSTANCES FOR LAZY	64

List of Figures

1.1	AN EXAMPLE BROADCASTING SYSTEM	6
1.2	AN EXAMPLE DATA BROADCAST SCHEDULING DIAGRAM	16
3.1	MINIMIZING TOTAL WAIT TIME WITH FCFS FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL	31
3.2	MINIMIZING TOTAL WAIT TIME WITH MRF FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL	32
3.3	MINIMIZING TOTAL WAIT TIME WITH AN ARBITRARY ALGORITHM A FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL	33
3.4	IMPACT OF INCREASING NUMBER OF REQUESTS ON TOTAL WAIT TIME, 10 REQUESTS PER TICK	46
3.5	IMPACT OF INCREASING NUMBER OF REQUESTS ON TOTAL WAIT TIME, 100 REQUESTS PER TICK	47
3.6	IMPACT OF INCREASING ARRIVAL RATE ON TOTAL WAIT TIME .	48
3.7	ALGORITHM PARAMETERS IN 3 DIMENSIONS - MRF	51
3.8	THE COMPUTATIONAL BURDEN OF TIE RESOLUTION	52

4.1	EXAMPLE SCHEDULING DIAGRAM OF LAZY	56
4.2	EXAMPLE SCHEDULING DIAGRAM OF GREEDY	57
4.3	EXAMPLE SCHEDULING DIAGRAM OF DYNAMIC PROGRAMMING ALGORITHM	63
4.4	BENCHMARKING AT BETA = 10	74
4.5	EFFECT OF INCREASING BETA	75
4.6	EFFECTS OF MODIFYING THE INPUT SEQUENCE	76
5.1	EXAMPLE SCHEDULING DIAGRAM OF MOST REQUESTS FIRST (MRF)	80
5.2	EXAMPLE SCHEDULING DIAGRAM OF GREEDY	82
5.3	BENCHMARKING AT BETA = 10	87
5.4	EFFECT OF INCREASING BETA to 30	88
5.5	EFFECTS OF MODELING THE INPUT SEQUENCE	89

ABSTRACT

Inherent in the field of data broadcasting is a communication problem in which a server is to transmit a subset of data items in response to requests received from clients. The intent of the server is to optimize metrics quantifying the quality of service the system provides. This method of data dissemination has proved to be an efficient means of delivering information in asymmetric environments demanding massive scalability. Of critical importance in such a system is the algorithm used by the server to construct a schedule of item broadcasts.

Due to the real-time nature of this problem, performances of heuristics designed to construct such schedules are heavily dependent on request instances. Thus it is challenging to establish the quality of one algorithm over another. Though several scheduling methods have been developed, these algorithms have been studied with a reliance on probabilistic assumptions and little emphasis on analytical results.

In contrast, we provide a formal treatment of the data broadcast scheduling problem in which analytical methods are applied, complemented by simulation experiments. Utilizing a worst-case technique known as competitive analysis, we establish bounds on the performance of various algorithms in the context of several different broadcast models. We describe results in three different settings.

Minimizing the total wait time of all requests with a single channel and multiple database items we establish the competitive ratios for two well-known algorithms, First Come First Served (FCFS) and Most Requests First (MRF) to be equal, and provide a general lower bound for all algorithms in this context. We describe simulation results that indicate the superior performance of MRF over FCFS on average. Minimizing two conflicting metrics, the total wait time and total broadcast cost, with a single channel and single database item we develop two on-line algorithms, establish their competitive ratios, and provide an optimal off-line algorithm used to simulate the impact of various parameters on the performance of both on-line heuristics. Finally, we extend the previous model by including multiple database items and establish a lower bound to a greedy algorithm for this context.

DATA BROADCAST SCHEDULING: MODELS, ALGORITHMS,
AND ANALYSIS

Chapter 1

Introduction

1.1 Motivation

Developments in delivery mechanisms and the rapidly growing demand for dissemination-based applications have created environments in which the method of data broadcasting has become increasingly popular. While technology improvements in mobile computing, satellite broadcasting, and cable networks have provided high bandwidth infrastructures, the amount of data that may be transmitted within these frameworks is much larger from server to clients than vice versa. Consequently, asymmetric properties of these systems prevent traditional unicasting techniques from being scalable or even usable [6, 8, 7].

At the same time, large data dissemination applications such as centralized databases, on-line auctions, and stock tickers are characterized by massive user populations requesting data from a centralized source. Capabilities of these services increasingly appear in non-traditional forms as evidenced by wireless PDA use by emergency responders and internet access for passengers on aircraft. Analogous to the infrastructures mentioned above, these services exhibit an imbalance, typically funneling most of the data to clients rather than from them. In addition, servers with large client bases see substantial redundancy in user requests

as many users desire the same content. Utilizing traditional techniques, a server running massive dissemination services will transmit the same information repetitively, responding to each client individually. This process simply does not scale and, as the number of both clients and requests increases, the burden on the server becomes immense. In contrast, data broadcasting is an efficient method of providing large-scale data delivery by simultaneously satisfying the needs of multiple clients.

Thus, the asymmetry of both modern infrastructures and application demands provides a synergistic setting for this technique that emphasizes data dissemination in one direction. The broadcasting method's resulting independence from the number of users in a system provides it with a scalability that has become increasingly attractive economically and academically.

Commercial use has already been made of the technique, including the Hughes DirectPC System, the Intel Intericast System, the Hybrid System, and the AirMedia System [8]. The applicability of the broadcasting method to cable television, mobile phones, and PDA services has come to fruition as well, with other applications forecasted [5].

Academic research in this area began in the 1980s in a series by Ammar and Wong [19] [10] [12] and has continued to present day. Of particular interest is the study of broadcast scheduling. Despite the advantages inherent in the broadcasting method, the bandwidth available to a server is still limited. Given the massive scale of systems utilizing data broadcast, the ability of a server to efficiently schedule the transmission of data to its clients is critical.

We are thus interested in the following communication problem. A single server is given access to a set of data items, channels over which the items may be transmitted, and requests

that arrive at various times for the items. At each system tick, the server must select an item to broadcast for each available channel, thus satisfying all outstanding requests for that item. The server will make these decisions through use of a single algorithm with the goal of optimizing some metric or metrics representing the quality of service the system provides. This problem can be differentiated into several model variations depending on the exact nature of the broadcasting system; the number and type of channels, database items, requests, and performance metrics are just a few parameters that impact the general nature and complexity of the problem. Regardless of any specific variation, the end result of each problem instance is the production of a schedule that describes which item has been broadcast to clients at each opportunity.

The question of which item should be broadcast at a particular time is a decision that the server must make immediately. In most real settings, a broadcasting server will not know the incoming request sequence beforehand. Thus the server must make ad hoc choices in determining which item to broadcast based on requests that have arrived. Algorithms designed to aid the server in this context are therefore operating with incomplete information and will provide approximate (heuristic) solutions. An algorithm that must perform an immediate action in response to partial information about the entire input sequence is known as an *on-line algorithm*. Of primary concern in such a setting is the evaluation of an on-line algorithm's effectiveness.

An ideal approach in determining the quality of an on-line algorithm is to assume a realistic distribution of requests and proceed with an analysis of average-case performance. However, assumptions concerning the arrival sequence of requests may require historical data that is not available and does not provide any absolute measure of an algorithm's

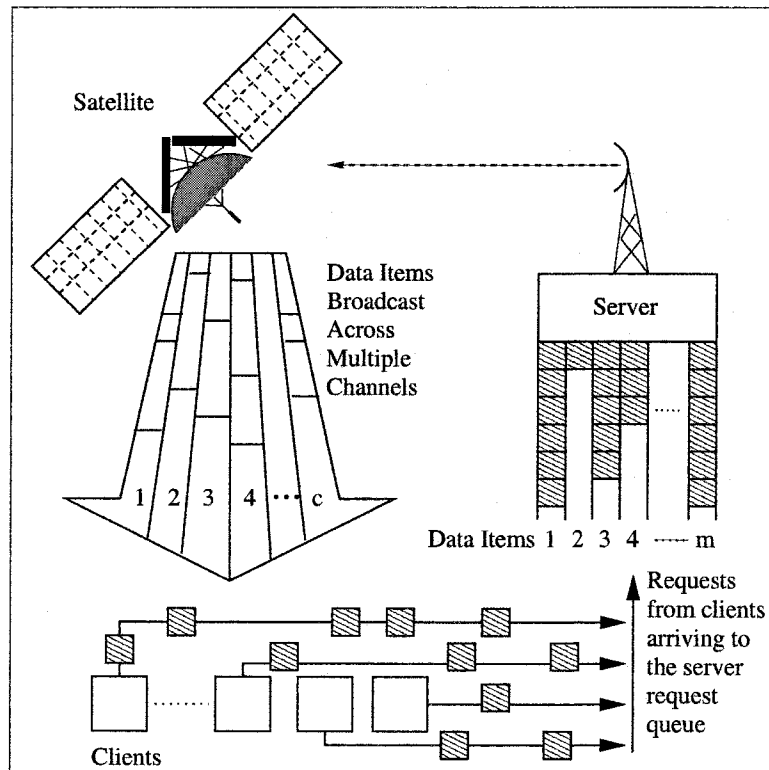
behavior over all possible scenarios. As an alternative method we take a comprehensive look at the problem using worst-case analysis techniques. While pessimistic, the appeal of this approach is the solidarity of performance measures independent from probabilistic assumptions. In addition, there are certain kinds of systems (such as emergency, financial, or martial) in which worst-case measures are critical. Though many algorithms have been developed for data broadcast scheduling, the work done to date has relied primarily on simulation or other distributional techniques to evaluate the performance of these algorithms with little emphasis on analytical results. The primary contributions of this thesis supply such results in a variety of contexts.

1.2 General Problem Description

Though the exact implementation of one data broadcasting system may differ from another there are several fundamental aspects that can be described. As an example system, consider Figure 1.1.

Primarily a broadcasting system will consist of a server that has access to a database of items and channels over which these items are transmitted. In Figure 1.1 we see a satellite broadcasting system in which the server is terrestrial but has access to multiple channels over which information is transmitted to terrestrial clients. This setup closely models actual commercial systems mentioned already, though of course many different physical implementations exist. Data items may be objects, web pages, or other information and it takes time to transmit them over a channel. Specifically, we define a system tick as the amount of time required to transmit an item of unit size. The number of items in the database is finite and

Figure 1.1: AN EXAMPLE BROADCASTING SYSTEM



static and we assume these items to be unit-sized, discrete, and independent of one another.

Each channel is accessible to every client and these clients are continually monitoring all channels for items that are of interest to them. In Figure 1.1, the satellite “bombards” terrestrial clients with its broadcasts. Every time a channel finishes transmitting an item (that is, the channel becomes available) the server will select another item to send over that channel. The scheduling decision of which item to send will be based primarily on the needs of its clients which are generating requests for particular items, though other information may be considered.

The system will respond to a sequence of requests, and not all requests may be serviced

immediately. At the beginning of each system tick, requests that have arrived from clients will be collected by the server and maintained in a queue. This queue is best conceptualized as a two dimensional structure¹. The one shown below the server in Figure 1.1 is collecting incoming requests, depicted as filled squares, being generated by the larger, open-squared clients. The horizontal dimension will be of length m establishing a “bin” for each distinct item. The vertical dimension will be of dynamic length representing the number of requests for a particular item collected in that item’s bin. The length of time a request remains in the queue is the wait time of that request. The server will consider (via its algorithm) requests in queue before deciding which items to send at the beginning of the next tick. Because each request will be in queue for at least one tick, the minimum wait time for any request is 1. Each time an item begins broadcasting, all requests for that item are removed from the queue. We can envision the bin for that item thus being reduced to zero length. Any request for the item that arrives after the broadcast will be placed in queue and must be satisfied by another broadcast of that item at a later time.

A schedule is a record of items broadcast during each tick over all ticks in the lifetime of the system. Typically, a system will continue to run on an input sequence until the server has satisfied every request in the sequence. Ultimately, the task of the server is to optimize one or more metrics representing the quality of service exhibited by the resulting broadcast schedule. To this end, each broadcast scheduling problem must supply an objective function that embodies the importance of each metric. (Typically, the objective is to minimize an

¹The queue structure may be simplified for many algorithms or contexts in which only the total number of waiting requests is relevant. However, in general there may be other characteristics of requests that require tracking them separately as is the case in the First Come First Served (FCFS) algorithm in which the arrival time of each requests must be recorded.

operational metric though an analogous maximization problem is feasible. We will restrict our discussion to problems of minimization.) The objective function is a mathematical definition that evaluates the effectiveness of a schedule in terms of the targeted metric(s). For example, if the targeted metric is total wait time, then the value of the objective function is the sum of wait times experienced by all requests. Most commonly, the server will attempt to minimize the total wait time of all requests as minimizing this value increases the quality of service to the clients.

Several other metrics are also applicable and the server may be called upon to attempt the minimization of more than one metric. In particular we consider objective functions in which each database item has an associated cost to broadcast that item. This cost is a general reflection of any strain performing a broadcast may cause to the system and can take many forms. The burden may be financial (paying a copyright fee per use), physical (the energy expenditure from the battery of a PDA), or even artificial (cost imposed to represent data priority). Minimizing this broadcast cost maximizes the quality of service to the server itself. Thus the majority of objective functions we consider attempt to minimize the sum of wait times experienced by all requests while simultaneously attempting to minimize the sum of all broadcast costs. Objective functions of this form are particularly interesting due to the conflicting nature of these two objectives.

Regardless of its form, the value of an objective function establishes a measure of how well one schedule does relative to another. Obviously for a minimization problem, we desire a schedule's evaluation via the objective function to be as small a value as possible.

Notice that because the wait time for any request is at least 1, we know that the sum of wait times for any possible schedule will be at least n , the number of requests in the input

sequence. Observations like this indicate that it is possible to make absolute statements about an algorithm's performance without considering the behavior of the algorithm itself. That is, we know that no algorithm is capable of constructing a schedule that will perform better than n in regards to minimizing total wait time. A comprehensive study of data broadcast scheduling requires that we be able to produce two kinds of evaluations relative to a problem: an evaluation of the best any algorithm can do and an evaluation of how well one specific algorithm can do relative to another known algorithm.

1.3 Formal Problem Definition

The general problem description given above establishes core characteristics of Data Broadcast Scheduling. Here we formally define the problem with the introduction of useful notation.

We are given a single server with access to a set of m distinct and unit-sized items. The server is in contact with an unknown (and irrelevant) number of clients that require these data items and will communicate their needs to the server instantaneously when they occur (any latency in information exchange with the server is ignored). Requests for items arrive to the server at various times and are placed into the server's queue. Each request j is characterized by at least two attributes: the arrival time of the request a_j and the item requested. We assume these items to be discrete and independent of one another such that the arrival of a request for one item yields no insight into the arrival of any other requests. Note that the number of requests n is finite.

The server has $c \geq 1$ channels over which items may be broadcast to all clients simul-

taneously in order to satisfy requests. These channels are dedicated (the server may use a channel's entire bandwidth) and perfect (no transmission errors will occur). All requests for item i that arrive before the server begins broadcasting i will be satisfied (removed from the queue) once the broadcasting of i has begun².

A tick at time t is the interval $[t, t + 1)$, defined as the amount of time required to broadcast one item over a single channel. Each tick a request remains in the queue it accumulates a unit of wait time. The total amount of wait time produced by request j is w_j . Without loss of generality, we assume that the system clock begins at time $t = 0$ and advances discretely until all items have been satisfied at time $t = z$.

A channel may only be used to transmit one item at a time. The item that begins broadcasting at time t on channel k is denoted b_{tk} . It is therefore possible that as many as the number of channels c and as few as 0 items will begin broadcasting each tick. The server incurs a cost β for each item broadcast (though it is possible $\beta = 0$).

For a given instance, the server will take its knowledge of the request sequence and consult its scheduling method to make each broadcasting decision. The scheduling method is an algorithm that will select, at every opportunity, which item should be sent on an available channel. The history of these decisions throughout the lifetime of the system constitutes a schedule S for the problem instance.

For each model there must be a standard by which the schedule produced for every problem instance is evaluated. A metric is a measurement under which a schedule will be evaluated. Though several observations may be made of a schedule, these measurements are

²The combination of perfect transmission and unit-sized items establishes that a request may be removed from queue once broadcasting begins. The removal of either assumption would require a different definition of request satisfaction.

deemed irrelevant if not reflected in the objective function for the problem. For example, if the objective function is designed solely for the minimization of the total wait time experienced by all requests, then the length of the schedule itself is not an issue. A schedule S_1 that minimizes the objective function but is longer than a schedule S_2 which does not is superior to S_2 . If the length of the schedule is important, it must be included in the objective function which may implicitly address many metrics simultaneously.

It is with the above backdrop that models may be established by specifying characteristics in three areas.

1.3.1 The Server's System Environment

Physically the system environment consists of the server, its database, and the channels over which the database items are transmitted. Details that describe the system hardware or its operation fall under this category. Several aspects which must be addressed are:

- **Server Knowledge:** A parameter that must be established for a problem instance is the extent to which the server has knowledge of the request sequence. A server operating in a real-time context will have no knowledge of incoming requests beyond what requests have already arrived. This is the on-line context. The opposite context, off-line, may also be considered in which the server has complete knowledge of the entire request sequence. Competitive analysis is the comparison of algorithms operating in these two contexts. The limitations of server knowledge (or lack thereof) must be specified before analysis of a problem may continue as this parameter may impact

the behavior of any algorithms used³.

- **Number of Channels and Items:** The number of channels in a system refers only to the bandwidth from server to clients (while present, bandwidth from clients to sever is ignored). These channels are identical, though indexing may be necessary to distinguish the utilization of one from another. The number of distinct items in the database is static throughout the problem. Each of the items in the server database are of unit size and are all associated with the same broadcast cost. If the broadcast cost of items is not presented, then the cost is assumed to be zero for all items. The number of channels and database items must be specified for each problem instance.
- **Scheduling Method:** A server will utilize an algorithm to make each scheduling decision. This algorithm will be affected by the extent to which the server has knowledge of the request sequence. For example, a brute-force algorithm operating on-line will not be able to consider requests that may arrive after the current system tick whereas its off-line counterpart can. Thus the on-line or off-line nature of each algorithm studied within a model must be established. Strictly speaking, an algorithm will also provide means through which a decision will be made in light of equally attractive choices. For example, an algorithm that chooses to satisfy the first arriving request (FCFS) will need to refine that decision if two requests arrive simultaneously for different items. While these tie-breaking decisions are typically not the focus of any analysis, instances do arise in which the tie-breaking method of an algorithm will

³Theoretically, there exist contexts between on-line and off-line in which the server has a limited ability to view some of the request sequence in advance. Algorithms with this “look-ahead” knowledge are known as semi-on-line.

impact its performance.

- **System Termination:** Without loss of generality, a system will begin with the clock set at $t = 0$. Time will advance discretely in the system until termination at time $t = z$. The criteria for system termination must be established for any problem instance. Typically, the system will terminate once all requests have arrived and have been satisfied⁴.

1.3.2 Request Characteristics

From time to time, clients will send requests to the server for a particular item. These requests are maintained in an “unsatisfied” status by the server in a queue. Each request j will be characterized by two attributes: 1) the arrival time a_j of the request and 2) the item requested $i \in 1 \dots m$. As previously stated, all requests for item i that arrive before the server begins broadcasting i will be satisfied (removed from the queue) once the broadcasting of i has begun. A request that is not satisfied by the broadcast of its item at the beginning of a tick will accumulate one unit of wait time and will continue to accumulate wait time in this manner. A request cannot be satisfied in the same tick in which it arrived; all requests will experience a minimum wait time of one unit.

The request sequence must be defined in full for a problem instance. Any specification describing the nature of a request fall under this category. Note that these attributes

⁴Note that the discrete event approach, the presence and advancement of the system clock, is an important distinction of the data broadcasting problem. Regardless of whether or not any new requests arrive during a system tick, the server’s scheduling algorithm will make a decision to broadcast an item as long as there are unsatisfied requests in queue. In contrast, a traditional on-line problem operating on an ordered sequence would “skip” periods of time from one request to another; each action will have been motivated by a request.

represent information not available to an on-line algorithm until the request has actually arrived to the queue. Some common request details are:

- **Arrival Time:** Each request will arrive at a specific point relative to the system clock. The request sequence will be ordered such that requests having the same arrival time will arrive simultaneously to the server queue⁵.
- **Item Requested and Satisfaction:** Each request will be for a particular item and will not be satisfied until that item is broadcast by the server. The transmission of the item requested is a necessary (but not always sufficient) condition for the satisfaction of the request⁶.
- **Deadlines and Penalties:** The deadline is the time by which a request must be satisfied or forever remain unsatisfied. Unless otherwise stated, the deadline for each request is assumed to be infinite.

1.3.3 The Objective Function

An objective function $OBJ()$ is required for each model. This function represents metric(s) under which a schedule S will be evaluated. Various algorithms used to construct a schedule will optimize the metric(s) at differing levels of performance. For our work, we are mainly interested in the following objective functions:

⁵Again, this arrival process is distinct from traditional request sequences. Permutations to the data broadcasting sequence will not change the problem instance as long as each request in the sequence has an arrival time less than or equal to the arrival time of the next request in the sequence.

⁶Unit-sized items are broadcast in a single tick. Systems allowing multiple sizes for items, however, face the broadcast of the item in “pieces” over multiple ticks. In such systems it is often allowable for the server to preempt a broadcast of one item before all pieces have been transmitted. Requests for preempted items are not satisfied in these systems until all pieces of the item have been broadcast.

$$OBJ(S) = \sum_{j=1}^n w_j + B \cdot \beta,$$

where the wait time of each request j , w_j , is summed over the total number of requests n and the total number of broadcasts B in schedule S is multiplied by the cost of broadcasting β .

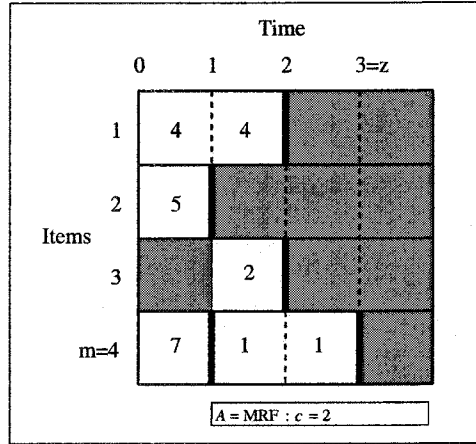
1.4 The Data Broadcasting Schedule Diagram

In order to describe the schedule constructed by an algorithm for a problem instance we have developed a diagramming technique. The Data Broadcasting Schedule Diagram (or Schedule Diagram for short) graphically depicts the arrival and satisfaction of requests as well as the broadcast of data items.

The diagram consists of grid that is m (the number of data items) tall by z (the lifetime of the system) wide. Each row is labeled with an item from the database and each column with a system tick. Each cell in the grid contains a value greater than 0. That value represents the number of requests for the item indicated by the index labeling the row. Those request are in queue at the time specified by the time indicated at the demarcation of each column. A cell that is grayed out indicates an absence of any requests for that item at the current tick. On the right side of each cell is a demarcation symbolizing any activity that occurred involving the requests for the item labeling that row. A dashed line indicates no change in the requests for that item. A thick solid line represents a broadcast for the row's item at the beginning of the column's tick. A thin solid line indicates a change in the

⁷ β is the normalized cost of broadcasting relative to a unit of wait time. Thus, a $\beta = 5$ implies that broadcasting is five times more expensive to a system than a single request waiting a single tick.

Figure 1.2: AN EXAMPLE DATA BROADCAST SCHEDULING DIAGRAM



number of requests waiting for the row's item. Specific instance information such as the algorithm used, the number of channels, and other specified parameters may be listed in a box below the diagram itself.

Consider the following example.

In Figure 1.2 we diagram a schedule constructed by the algorithm Most Requests First (MRF). MRF is an algorithm that will broadcast the item with the most outstanding requests. The schedule is constructed in response to a trivial input sequence for two channels. An indication of the scheduling algorithm and the number of channels can be found in the bottom right corner of the diagram.

The arrival sequence is as follows. At time $t = 0$ we see the arrival of four requests for item 1, five requests for item 2, and seven requests for item 4. At time $t = 1$ we see that the server broadcasts items 2 and 4. Note that while no additional requests arrived at $t = 1$ for item 2, a request did arrive for item 4. Because this request arrived after the broadcast of item 4, the request is queued. We also see that items with outstanding requests were not broadcast due to the limiting number of channels in this system, $c = 2$. Because no change

occurred in the number of requests for item 1, it is demarcated by a dashed line while two additional requests arriving for item 3 causes the solid line to be placed on that row.

At $t = 2$ the server again broadcasts two items, 1 and 3, both of which are marked by the thick solid line. The remaining two items, 2 and 4, experience no change and are marked by the dashed line. Finally, at the completion of the schedule at $t = 3$ we see that only one item, 4, is sent as there are no other outstanding requests.

The scheduling diagram also allows the objective function evaluation of a schedule to be derived graphically. Consider again the objective function that incorporates total wait time and total broadcast cost:

$$OBJ(S) = \sum_{j=1}^n w_j + B \cdot \beta.$$

The sum of wait times experienced by all requests over the length of the schedule $\sum_{j=1}^n w_j$ can be determined graphically by summing the numerical values of all cells in the diagram. In the above example, this sum is 24. The total broadcast cost $B \cdot \beta$ is graphically determined by counting the number of thick solid lines and multiplying that number by the broadcast cost β . Assuming $\beta = 5$ in the above example we see the total broadcast cost of the schedule to be 25. Visually, then, we can confirm the objective function evaluation of this example schedule to be 49.

This diagram will be used throughout the remainder of this thesis. Its strength lies in the ability to succinctly represent the behavior of an algorithm on a given input sequence in a variety of models.

1.5 Thesis Organization and Contributions

We organize the remainder of the thesis as follows. In Chapter 2 we provide a preliminary background on Competitive Analysis (our primary analysis technique) and describe the two fundamental system environments (push and pull) that dichotomize the field of broadcast scheduling. Chapters 3, 4, and 5 describe our results in three different settings. In Chapter 3 we consider the popular model of minimizing the total wait time of all requests in the context of a single channel and multiple database items. We establish competitive ratios for two well-known algorithms, First Come First Served (FCFS) and Most Requests First (MRF), and provide a general lower bound for all algorithms in this context. Chapter 4 introduces the concept of broadcast cost and considers a model in which the server attempts to minimize both the total wait time and total broadcast cost in the context of a single channel and single database item. To our knowledge, this dual metric problem has never been applied to the pull-based system environment. We develop two algorithms (LAZY and GREEDY) and establish their competitive ratios. In addition, we develop an optimal off-line algorithm for this context. Chapter 5 extends the model of Chapter 4 by including multiple database items. Within this context we establish a lower bound for the GREEDY algorithm. All three chapters provide a backdrop of simulation experiments as a complement to the aforementioned analytical results. We conclude in Chapter 6 and describe future research directions.

Chapter 2

Related Work

2.1 On-Line Algorithms and Competitive Analysis

In order to study the algorithms in our work we primarily employ a technique known as competitive analysis. In this section, we review the the characteristics of on-line algorithms and competitive analysis that motivated the use of this worst-case method. An interesting difficulty arises when one begins to compare the performance of two or more sub-optimal (heuristic) algorithms attempting to schedule the allocation of resources to actions taken over time. Heuristics designed to solve these problems will operate at various levels of quality depending on the input sequence. An algorithm that performs quite well with one sequence may appear to act foolishly in another situation. Thus, establishing the quality of any particular algorithm A_1 over another A_2 becomes problematic in the absence of distributional assumptions; one can construct sequences in which the schedules produced by A_1 are superior to those produced by A_2 , inferior, or even equal.

The difficulty in comparison originates in the dependence of each problem on an input sequence that is connected to timed events. The input sequence for the data broadcast scheduling problem is the arrival sequence of requests. Each request arrives at the beginning

of a specific system tick with information on, among other things, the item its client needs. Because this sequence is time-related, an algorithm cannot choose to schedule an action until the input motivating that action has come to pass. In our problem, the server cannot choose to satisfy a request that has not yet arrived.

Traditionally, algorithms acting on ordered sequences are divided into two broad categories: on-line and off-line. An *on-line* algorithm is one that does not have any knowledge of the input sequence beyond what has already occurred and must perform an action in response to each request that arrives [32]. In contrast, an *off-line* algorithm is one that is given the entire input sequence *a priori*. Though it cannot act on events that have not yet occurred, it can act in anticipation of events it knows will come to pass. Obviously, we expect in most instances that an off-line algorithm will have a distinct advantage over its on-line counterpart.

We note that the traditional definition of an on-line algorithm is, strictly speaking, slightly different from the data broadcast scheduling problem we present. In the traditional definition, an input sequence is typically supplied one request at a time and is organized by the order of requests [41]. However, such a categorization can be easily extended to problems involving timed sequences. The data broadcast scheduling problem involves a sequence of requests organized by the arrival time of each request and these arrival times are related to the system clock embedded in the problem. Thus, at the beginning of each system tick the on-line algorithm will be presented with zero or more requests and must make an immediate decision based on, not necessarily one, but potentially many requests that have arrived. The distinction to be made, then, is that any permutation in the order of requests in a traditional input sequence would constitute a different instance whereas our

use of the input sequence allows for permutations in the order of requests with the same arrival time without changing the instance. In either situation, the off-line algorithm is still given the entire sequence *a priori*.

If the comparison of two on-line algorithms is difficult, then there are serious implications to a discussion of an on-line algorithm's optimality. Clearly, the optimality of an off-line algorithm is straightforward. An optimal off-line algorithm is one that performs at least as well as any other off-line algorithm for all possible instances. We can say with certainty that an exhaustive search will provide such an optimal solution. However, for any on-line algorithm we can construct a sequence for which that algorithm fails to perform as well as another on-line algorithm we select. What, then, is the criteria to establish an on-line algorithm as optimal?

To address this question we turn to a worst-case analysis technique known as competitive analysis [32]. Early versions of competitive analysis grew from a combination of worst-case assumptions, cost amortization [23], and comparison to off-line algorithms used by Bentley and McGeoch to study the list accessing problem [16, 36]. The technique as a general approach was advocated by Sleator and Tarjan [44], ultimately named by Karlin et. al. [31], and formalized by Manasse et. al. [36]. Competitive analysis establishes the quality of an on-line algorithm by comparing its performance to that of an optimal off-line algorithm and taking the worst case for the same input sequence. When evaluated in this manner over all possible sequences, a competitive ratio is derived that establishes a measure by which the on-line algorithm may be evaluated. Specifically, an on-line algorithm with the smallest competitive ratio possible is said to be optimal (in the worst case). Though pessimistic as any worst-case analysis, the appeal to this approach is the absence of any distributional

assumptions.

Formally, for a given problem we consider an on-line algorithm A , the optimal off-line algorithm OPT , and any instance of the problem I . We define $A(I)$ and $OPT(I)$ to be objective function evaluation of the on-line and optimal off-line algorithms respectively. Algorithm A is said to be r -competitive if for all possible finite instances I :

$$A(I) \leq R \cdot OPT(I) + \alpha,$$

where α is a constant independent of I ¹. The value R is called the competitive ratio of A in the context of the problem. Though R may depend on the parameters to the problem, it must remain independent of the input I . An R -competitive algorithm A is therefore guaranteed to perform within a factor R of the optimal off-line algorithm OPT , allowing for a bit of leeway α . When $\alpha = 0$ we say that A is strictly R -competitive.

An intuitive way of viewing the use of competitive analysis is to relate the technique to a game between an on-line player and a “malicious”, all-knowing adversary [17]. The on-line player is attempting to solve the problem in question by running an on-line algorithm on an instance the adversary provides. The adversary is malicious in that its intent is to maximize the competitive ratio. This involves making the on-line algorithm perform as poorly as possible while simultaneously favoring the performance of the off-line algorithm. The technique is worst-case because the adversary has the advantage of knowing exactly how the on-line algorithm behaves and will therefore be able to construct the worst possible instance².

¹More specifically, the competitive ratio R is defined to be the infimum over the set of all values R .

²This discussion is for deterministic algorithms only. There are more exotic variations of competitive

Often, the approach to establishing a competitive ratio for any particular algorithm is to provide a bound of that algorithm's performance relative to the optimal from above and below. A proof that the upper bound and lower bound are equal will establish a tight ratio. A general bound is one in which the algorithm used in the proof remains arbitrary with no assumptions about its behavior. As such, the ratio obtained will hold for all possible algorithms in the context of the problem. General bounds are naturally more difficult to provide. A general lower bound is typically derived through use of an adversary argument; one needs only to establish an example instance that produces the lower bound. Proving a meaningful general upper bound is considerably more difficult since it involves proving that the competitive ratio is no larger than r for all possible instances and all possible on-line algorithms [32].

While its roots grew from combinatorial optimization, competitive analysis has been applied to a wide variety of problems in several fields, not all of them theoretically focused. Many financial or martial problems for example are inherently on-line and favor the pessimistic nature of competitive analysis.

2.2 Data Broadcasting System Environments

Some of the earliest research to address the problem of data broadcast scheduling is a series of works by Ammar and Wong [10] [11] [19] [48] from the 1980s. They study various aspects of the Teletext³ system and present heuristics in which schedules are constructed *a priori*

analysis in which the adversary is limited in its knowledge and abilities such as the diffuse adversary of Koutsoupias and Papadimitriou [35]. Other variations such as loose competitive analysis developed by Young [50, 49, 51, 52], or the use of randomized algorithms [35, 52, 40] have also been proposed.

³Developed in the 1970s, TELETEXT is a one-way system for transmission of text and graphics via broadcasting or cable for display on a television set. While some teletext systems are still in use they have

based upon assumed arrival rates. Among their contributions are the derivation of a lower bound on the average waiting time for users of a Teletext system and the establishment of optimal schedule characteristics when arrival distributions are known (or assumed).

The concept of Broadcast Disks, introduced by Aksoy and Franklin et al. [3] [2] [4] [14] generalizes the problem to an arbitrary number of channels using a shared communication medium for information distribution in asynchronous settings. Their work addresses both interactive and non-interactive systems, with the key problem in all settings being the scheduling of data transmission. In addition to the heuristics of Ammar and Wong [10] [19] a few well-known algorithms for the problem have been studied more recently through simulation by Aksoy and Franklin [8] [7] [2]. These include: First Come First Served (FCFS) that broadcasts items in the order they are requested, Most Requests First (MRF) that broadcasts the current “most popular” item, and Longest Wait First (LWF) that sends the item with the largest cumulative wait time over all requests for that item. These algorithms are all somewhat naive though experimental results in the above studies indicate that LWF performs quite well in the minimization of total wait time. The decision overhead of LWF, however, prohibits its practical implementation motivating the development of an approximation to LWF by Ulusoy in [45] and the development of a non-naive and adaptive algorithm called RxW by Aksoy and Franklin [8] [7] [6]. RxW algorithm has parameters that may be adjusted for performance that is better than its naive counterparts under various scenarios⁴ including a trade-off between a focus on average and worst case wait time⁵. The performance evaluation of RxW, as in most of the evaluations performed in the

largely been replaced by interactive videotext systems, computer-based interactive systems, of which the Internet is an aggregate example.

⁴RxW has proved to be applicable to scheduling issues in the supportive field of data staging [9].

⁵Despite the superior performance of RxW it can also be expensive to implement, a problem addressed

above research, was carried out through simulation experiments.

In contrast, Kalyanasundaram and K. Pruhs et. al. [30] [29] [42] present an entirely analytical study of the algorithms mentioned above using a worst-case technique known as resource augmentation analysis. In this approach, the performance of an on-line algorithm with a certain amount of resources is compared to an off-line algorithm with fewer resources. While analytically similar to the worst-case analysis we employ (competitive analysis), the work differs from our research in the comparison of an on-line algorithm with more channels to an off-line algorithm with fewer channels, while we compare algorithms operating with the same resources available. They also make no consideration of broadcast costs. True competitive analysis, while suggested by Goldberg et. al. [22], was not undertaken until Mao [37] and later by Hawkins and Mao [27].

The differing characteristics of broadcasting in asymmetric environments are discussed at length by Aksoy et. al. in [6]. Fundamentally, there are two settings in which broadcasting is done, dichotomized by the use of client requests and referred to either as “pull” or “push”. The pull-based version of the problem is the classic setting in which clients communicate their requests explicitly to the server. The model, as we have described it, is pull-based and we operate exclusively within this framework. The terminology derives from the event-driven nature of a server in a pull-based system. The clients, by asking the server for specific information, will “pull” that information to them. In the absence of requests the server typically will not disseminate items at all.

In contrast, push-based broadcasting involves clients that do not, for whatever reason, communicate requests explicitly to the server. As such, the server must rely on historic

by Goldberg et. al. in [22].

data, distributional assumptions, or predictive methods to anticipate what those requests may be. Push methods rely heavily on stochastic models and periodic schedules made *a priori*. There are implementation advantages and disadvantages to both the pull and push based models that are worth noting.

Like the early Teletext system, some environments are inherently push-based⁶. This version is common in mobile computing, especially in settings in which there is no bandwidth at all from clients to server. If communication is possible in both directions, however, the extent to which one setting is favored over the other will be impacted heavily by the cost of communicating with the server and by the longevity of items in the database. A pull-based system will require a significant amount of client-to-server communication and its database will be virtually static; the clients must know the exact content of the database to make explicit requests. Updates to database content are possible but will incur a high cost of periodic client polling. Push-based systems avoid both concerns. No communication with the server is done at all and clients need not know the database content. Because the server is “pushing” unsolicited information out to its clients without feedback however, there is a risk of wasting resources on irrelevant items. In addition, it is possible that a client never receives the data it requires. Thus, the effectiveness of a push-based system will depend on how accurately the server anticipates the “hidden” needs of its clients. This anticipation is studied predominantly through stochastics by assuming (most commonly) a Poisson or Zipf distribution of client requests.

Vaidya and Hameed have proposed several algorithms used to construct push-based

⁶Although teletext may appear to the viewer to be interactive, it is not. When one punches in an item number on a teletext decoder, the machine simply waits for that page to be broadcast, captures it, and displays it on the television set. No request is sent out from the machine to the information server.

schedules. In [38, 46, 25, 26] their work considers the development of algorithms with the intent of minimizing wait time, perhaps the most commonly utilized metric. Several model variations are considered including environments involving multiple channels and transmission errors. The performance of the algorithms is evaluated under these differing assumptions using simulation and some analysis. Jiang and Vaidya consider the minimization of response time variance in [28] as an alternative metric. Again, algorithms are both developed and analyzed using a combination of simulation and time complexity analysis. Additionally, some of these algorithms can be adapted to the pull-based version of the problem. Other interesting metrics such as minimizing server response time [47], minimizing operation costs [13], and minimizing the maximum response time [15] have also been proposed.

Considering yet another metric in the push-based model, Bar-Noy et. al. [13] and Kenyon, Schabanel, and Young [34] present efficient schemes for minimizing schedule cost. Interestingly, this metric is actually the combination of response time and the cost incurred by a server for item transmission. The concept inspired our development of objective functions with multiple minimization criteria discussed in Chapters 4 and 5. Kenyon and Schabanel continue with this metric in the context of a database with non-uniform item sizes [33] and Schabanel singly in the context of preemption [43].

The complexity⁷ of the push-based data broadcast problem has been well-studied. In [13], Bar-Noy et. al. establish the NP-hardness of the push-based model in the context of minimizing response time and broadcast cost. Schabanel continues to define the NP-hardness of the problem when it is expanded to include non-uniform item sizes and pre-

⁷For a complete treatment of NP-hardness see Garey and Johnson [21].

emption of broadcasts. Only recently has the complexity of the pull-based model been established. In [20], Erlebach and Hall provide an NP-hardness proof for the broadcast scheduling problem in the context of minimizing wait time with unit-sized items and any number of channels, the model we address in Chapter 3.

Potentially, a pull-based system is able to achieve a better performance than a push-based one, at the cost of additional bandwidth from clients to server. While early broadcasting research tended to focus on push-based systems, the increasing availability of bandwidth from clients to server has recently fueled increasing interest in pull-based models. In addition, hybrid architectures that utilize characteristics of both push and pull-based systems have been suggested by Acharya et. al [1], Oh et. al. [39], and Guo et. al. [24]. Despite the continuing appearance of research in this field we see that studies through experimentation are the more populous approach.

Chapter 3

Minimizing Total Wait Time: The Multi-Channel Case

We now consider the Data Broadcast Scheduling Problem in which the single goal of the server is to maximize the quality of service the system provides to its clients. In this chapter we present both analytical and simulation results regarding algorithms that attempt to optimize this metric by producing a schedule that will minimize the total wait time of all requests. Though this is the most common optimization function found in the field, the use of competitive analysis to study this model was not employed until Mao [37] whose work we extend from its original single channel restriction¹. This chapter first formally defines the model and then proceeds to describe the analytical and simulation results of two on-line algorithms obtained within its context.

¹A significant portion of this chapter is published in Hawkins and Mao[27]

3.1 Model Description

Building upon the general problem definition established in Section 1.3, we are given a server with access to a database of m unit-sized items and c channels over which the items may be transmitted. A sequence of requests is also given where each request arrives for some item. It is possible, therefore, that the server will make as many as c decisions concerning which items should be broadcast at the beginning of each system tick. In order to maximize the quality of service the system provides to its clients, the server will attempt to minimize the wait time experienced by all requests. That is, an evaluation of the schedule S produced in this model will be determined by the objective function

$$OBJ(S) = \sum_{j=1}^n w_j,$$

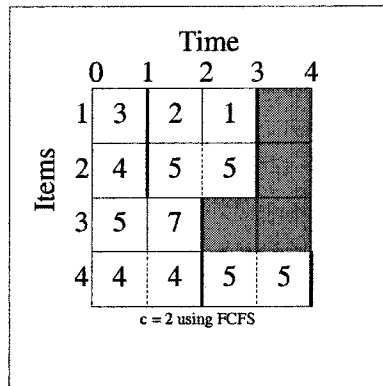
where the wait time of each request j , w_j , is summed over the total number of requests n .

3.2 Algorithms

We consider two well-known on-line algorithms the server may employ to minimize the above objective function, First Come First Served (FCFS) and Most Requests First (MRF). Though these algorithms are somewhat naive, we show their performance in the worst case to be as good as any algorithm can hope to achieve.

First Come First Served (FCFS) selects for broadcast the item that will satisfy the request with the earliest arrival time. In a sense, this algorithm attempts to prevent request starvation and, in so doing, minimize the number of requests with very long wait times. When several requests all have the earliest arrival time, a tie-breaking mechanism must be

Figure 3.1: MINIMIZING TOTAL WAIT TIME WITH FCFS FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL



specified. As an example, consider a trivial system with $c = 2$ and $m = 4$ employing FCFS as its scheduling algorithm. Figure 3.1 displays the resulting schedule when this system is faced with the following requests sequence. At $t = 0$ requests arrive for all items: three requests for item 1, 4 requests for item 2, five requests for item 3, and four requests for item 4. FCFS, having four equally good candidates, will choose the first two items for broadcast. At $t = 1$ only the first three items see additional requests: two requests arrive for item 1, five requests for item 2, and two requests for item 3. Because two items still contain requests arrived from last tick, FCFS will select items 3 and 4 for broadcast. Finally, at $t = 2$ the last requests, one for item 1 and five for item 4, arrive. Not having addressed the requests arrived from the previous tick, FCFS selects items 1 and 2 for broadcast. We see that FCFS is somewhat behind and must use one additional tick after all requests have arrived to broadcast item 4 one last time. From the diagram we can compute the objective function evaluation of this schedule by summing all requests left unsatisfied each tick. We see that FCFS has produced a schedule with a total wait time of 50 for this instance.

Figure 3.2: MINIMIZING TOTAL WAIT TIME WITH MRF FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL

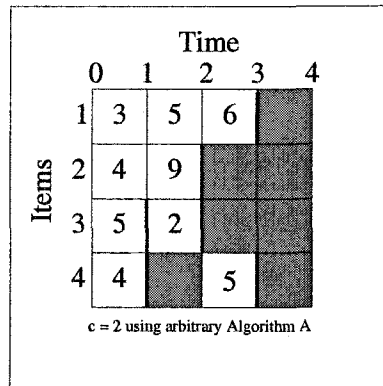
		Time				
		0	1	2	3	4
Items	1	3	5	1	1	
	2	4	5			
	3	5	2	2		
	4	4	4	9		

c = 2 using MRF

As a different technique, Most Requests First (MRF) selects for broadcast the item having the largest number of unsatisfied requests currently in queue. This algorithm takes a more direct approach to minimizing the objective function by attempting to minimize the wait time for as many requests as possible each tick. While, in general, we would expect this strategy to be superior to that of FCFS, the relative weakness of any on-line algorithm means this superiority cannot be guaranteed. Consider the performance of MRF when applied to the same example system and input instance given to FCFS above. The schedule produced by MRF on this instance is shown in Figure 3.2. At the first opportunity MRF selects item 3 for broadcast as it has the largest number of requests waiting, and item 2 since it is the lower index of the two items having the second largest number of waiting requests. MRF will continue to select for broadcast as many as two items with the largest number of requests at each opportunity until all requests are satisfied. It can be seen that the schedule produced by MRF for this instance has a total wait time of 45.

It should be noted that neither FCFS nor MRF have constructed the best schedule

Figure 3.3: MINIMIZING TOTAL WAIT TIME WITH AN ARBITRARY ALGORITHM A FOR THE MULTI-ITEM, MULTI-CHANNEL MODEL



possible for the given instance. Consider that some on-line algorithm A could provide a superior solution with total wait time of 44 as shown in Figure 3.3. We know that a brute force algorithm operating offline with knowledge of the input sequence in advance would also construct this schedule.

3.3 Analytical Results

As stated earlier, the wait time experienced by any request j for a unit-sized item is $w_j \geq 1$. Thus the minimum total wait time of any schedule independent of the number of channels or the scheduling algorithm is at least n . In this section we provide competitive ratios for FCFS and MRF and a general lower bound for all algorithms minimizing total wait time for the multi-channel, multi-item context.

3.3.1 First Come First Served (FCFS)

We define $\sum_{j=1}^n w_j(FCFS)$ to be the total wait time of the schedule given by FCFS for any instance. Likewise, $\sum_{j=1}^n w_j(OPT)$ is the total wait time of the optimal solution for the same instance.

Theorem 3.1 $\sum_{j=1}^n w_j(FCFS) \leq \frac{m}{d} \sum_{j=1}^n w_j(OPT)$, where $d = \min(c, m)$.

Proof: As already stated, $\sum_{j=1}^n w_j(OPT) \geq n$. Let $d = \min(c, m)$. We wish to show $w_j \leq \frac{m}{d}$ for $1, \dots, n$ in the FCFS solution to derive

$$\sum_{j=1}^n w_j(FCFS) \leq \frac{m}{d} n \leq \frac{m}{d} \sum_{j=1}^n w_j(OPT).$$

Case 1. $m \leq c$: If the number of items is less than or equal to the number of channels, then it is possible to broadcast all items each tick. That is, all requests are satisfied with the minimum wait time of 1 and the total wait time of the solution provided by FCFS is equal to that of the optimal solution. Since $m \leq c$ then $d = \min(c, m) = m$ and $\sum_{j=1}^n w_j(FCFS) = \sum_{j=1}^n w_j(OPT)$.

Case 2. $m > c$: For contradiction, assume there exists some request j for item i_1 with $w_j > \frac{m}{d}$. That is, the request arrives at time a_j and is satisfied at time $a_j + w_j$. Since $w_j > \frac{m}{c}$, we know that item i_1 is not broadcast between $a_j + 1$ and $a_j + \lceil \frac{m}{c} \rceil$. Thus, there remain $m - 1$ items to be broadcast during the $\lceil \frac{m}{c} \rceil$ ticks between $a_j + 1$ and $a_j + \lceil \frac{m}{c} \rceil$, $[a_j + 1, a_j + 2), \dots, [a_j + \lceil \frac{m}{c} \rceil + 1)$. We see that none of the c channels are idle during this period, else item i_1 would have been selected for broadcasting. It then follows that there

have been a total of $c \cdot \lceil \frac{m}{c} \rceil$ items broadcast during this period. Given that $m > c$ we know $c = \min(c, m) = c$, resulting in $c \cdot \lceil \frac{m}{c} \rceil \geq m$ items broadcast out of $m - 1$ items. By the pigeonhole principle, at least one item, say i_2 , among the $m - 1$ items must have been broadcast more than once during the interval. This is in contradiction to the FCFS algorithm; request j arriving at a_j requires item i_1 to be selected for broadcast at least once before item i_2 is selected a second time. We conclude that all requests have wait times no larger than $\frac{m}{c}$ in any FCFS solution. ■

Theorem 3.2 *There is at least an instance for which $\sum_{j=1}^n w_j(\text{FCFS}) = \frac{m}{d} \sum_{j=1}^n w_j(\text{OPT})$, where $d = \min(c, m)$.*

Proof: Consider the following instance with $n = xm$ for some x and $m > c$ (We do not illustrate the trivial case when $m < c$). Without loss of generality, take c, m such that $c|m$. Assume that for request j the arrival time is $a_j = 0$ for the first m requests and the requested items are $1, \dots, m$ respectively. For each successive request $j = kc + y + m$ where $y = 1, \dots, c$ and $k = 0, \dots, \frac{m(x-1)}{c}$ the arrival time is $a_j = k + 1$ and the requested item is $(kc + y + m) - m \cdot \lfloor \frac{kc+y+m}{m} \rfloor$. The FCFS algorithm gives a schedule that selects items in a circular order of $1, 2, \dots, m$ and broadcasts them in groups of c until all n requests are satisfied. Thus the FCFS algorithm creates a schedule in which for each item there is exactly one request waiting at any time $t : 1 < t \leq (\frac{n-m}{c})$. After time $t = (\frac{n-m}{c})$ no more requests will arrive and $m - c$ requests will remain to be satisfied. These remaining requests will be satisfied in $\frac{m-c}{c}$ additional ticks. That is, the schedule produced by FCFS will be completed at the end of tick $t = (\frac{n-m}{c}) + \frac{m-c}{c} = \frac{n-c}{c} = \frac{n-c}{c}$. Specifically, the c items broadcast at time $t = 0$ are $b_{01} = 1, \dots, b_{0c} = c$ where b_{tc} is the index of the item broadcast

at time t on channel c . For $t > 0$ the items broadcast at time t are $b_{ty} = tc + y - m \cdot \left\lfloor \frac{tc+y-1}{m} \right\rfloor$ for $y = 1, \dots, c$ and $1 \leq t \leq t = \frac{n-c}{c}$

Considering the total wait time of this schedule, we notice that the wait time of each of the first c requests is $w_j = 1$. The next c requests experience 2 units of wait time and so on. Thus the total wait time of the first m requests is $\sum_{j=1}^m cj$. Because it takes $\frac{m}{c}$ ticks to satisfy the last of the initial m requests, requests $m + 1, \dots, m + c$ will each experience a wait of $w_j = \frac{m}{c}$. By the time the $m + c$ request is satisfied, requests $m + c + 1, \dots, m + 2c$ will also have experienced a wait time of $\frac{m}{c}$. The total wait time of satisfying the remaining $n - m$ requests is then $(n - m)\frac{m}{c}$. We now have the total wait time of the schedule produced by FCFS:

$$\sum_{j=1}^n w_j(FCFS) = \sum_{j=1}^m cj + (n - m)\frac{m}{c} = \frac{(c + m)m}{2c} + (n - m)\frac{m}{c}.$$

The optimal schedule will produce unit wait time for most requests. Specifically, the first c requests satisfied will experience $w_j = 1$. The optimal schedule is then constructed such that the wait time experienced by each request arriving after the m th request has a wait time of $w_j = 1$. The total wait time for all but the first m requests is then $n - m$. Of the first m requests to arrive, the first c requests will experience a wait of 1, that is $w_j = 1$ for $j = 1 \dots c$. The second c requests will experience a wait of 2, that is $w_j = 2$ for $j = c + 1 \dots 2c$ and so on. Thus the total wait time for the initial m requests is $\sum_{j=1}^m cj$. Therefore, the total wait time accumulated by the optimal schedule is

$$\sum_{j=1}^n w_j(OPT) = \sum_{j=1}^{\frac{m}{c}} cj + (n - m) = \frac{(c + m)m}{2c} + (n - m).$$

Thus, for fixed m, c and arbitrarily large n , the ratio of $\sum_{j=1}^n w_j(FCFS)$ over $\sum_{j=1}^n w_j(OPT)$ approaches $\frac{m}{c}$. ■

3.3.2 Most Requested First (MRF)

We define $\sum_{j=1}^n w_j(MRF)$ to be the total wait time of the schedule given by MRF for any instance. Likewise, $\sum_{j=1}^n w_j(OPT)$ is the total wait time of the optimal solution for the same instance.

Theorem 3.3 $\sum_{j=1}^n w_j(MRF) \leq \frac{m}{d} \sum_{j=1}^n w_j(OPT)$.

Proof: Case 1. $m \leq c$: If the number of items is less than or equal to the number of channels, then it is possible to broadcast all items each tick. That is, all requests will be satisfied with the minimum wait time of 1 and the total wait time of the solution provided by *MRF* will be equal to that of the optimal solution. Since $m \leq c$ then $d = \min(c, m) = m$ and $\sum_{j=1}^n w_j(MRF) = \sum_{j=1}^n w_j(OPT)$.

Case 2. $m > c$: Let A_t be the number of requests arriving at t . Let B_{tk} be the number of requests satisfied by the broadcast at t on the k th channel, which is for item b_{tk} for $k = 1, \dots, c$. Let W_t be the number of requests still waiting at t . Let z be the last broadcast tick in the MRF schedule. We note that $W_z = 0, A_z = 0, B_{zk} = 0$ for all k , and $\sum_{t=1}^z (\sum_{k=1}^c B_{tk}) = \sum_{t=0}^{z-1} A_t = n$. Since MRF always chooses the item requested most, B_{tk}

is at least as large as the number of requests for any item other than item $b_{tk}, \forall k$. Thus the average number of requests satisfied by an item broadcast at time t , $\frac{1}{c} \sum_{k=1}^c B_{tk}$ is also at least as large as the number of requests for each of the $m - c$ items waiting but not broadcast at time t . So $\frac{m-c}{c} \sum_{k=1}^c B_{tk} \geq W_t$. We have

$$\sum_{t=1}^{z-1} W_t \leq \frac{m-c}{c} \sum_{t=1}^{z-1} \left(\sum_{k=1}^c B_{tk} \right) < \frac{m-c}{c} \sum_{t=1}^z \left(\sum_{k=1}^c B_{tk} \right) = \frac{m-c}{c} n.$$

Consider the total wait time $\sum_{j=1}^n w_j(MRF)$. A_0 is the amount of wait time accumulated during $[0, 1)$. $W_1 + A_1$ is the amount of wait time accumulated during $[1, 2)$. $W_2 + A_2$ is the amount of wait time accumulated during $[2, 3)$. And finally, $W_{z-1} + A_{z-1}$ is the amount of wait time accumulated during $[z-1, z)$. So we have

$$\begin{aligned} \sum_{j=1}^n w_j(MRF) &= A_0 + (W_1 + A_1) + (W_2 + A_2) + \cdots + (W_{z-1} + A_{z-1}) \\ &= \sum_{t=0}^{z-1} A_t + \sum_{t=1}^{z-1} W_t \\ &< n + \frac{(m-c)}{c} n \\ &= \frac{mn}{c} \end{aligned}$$

Together with $\sum_{j=1}^n w_j(OPT) \geq n$ for the optimal algorithm, we get

$$\sum_{j=1}^n w_j(MRF) \leq \frac{m}{c} \sum_{j=1}^n w_j(OPT).$$

■

Theorem 3.4 *There is at least one instance for which $\sum_{j=1}^n w_j(MRF) = \frac{m}{d} \sum_{j=1}^n w_j(OPT)$.*

Proof: We see that the same instance used for the earlier FCFS proof can be used to yield the same result for the MRF algorithm. Note that when MRF is applied to that instance, every broadcast decision $k = 1, \dots, c$ is made breaking an $(m - k)$ -way tie in favor of the item requested by the earliest arriving request. ■

3.3.3 General Lower Bound for c Channels

In this section we will prove that no on-line algorithm for broadcast scheduling with c channels and m unit-sized items has a competitive ratio better than $\frac{m}{d}$, where d is the minimum between the number of channels c and the number of items m .

Theorem 3.5 *For any on-line algorithm for broadcast scheduling with c channels and m unit-sized items, its competitive ratio is at least $\frac{m}{d}$, where $d = \min(c, m)$.*

Proof: We use an adversary argument and assume that the input instance is provided by the adversary. The arbitrary scheduling algorithm we consider is *on-line*; it has no knowledge of the request sequence except for the requests that have already arrived by the time each scheduling decision must be made. Because of this limitation, the adversary is able to wait until the algorithm has made its scheduling decisions each broadcast tick before deciding what items to request for the upcoming tick. The adversary has complete knowledge of the inner workings of the algorithm and will seek to provide the most challenging input instance.

The adversary initially makes m requests with arrival times of $a_j = 0$ and one request for each item. The on-line algorithm will select some or all of these items to broadcast at time $t = 1$ depending on the number of channels c relative to the size of m . We refer to the

item broadcast at time t on channel c as b_{tc} . Thus, the algorithm will select $d = \min(c, m)$ items to broadcast each tick, and on the first tick this will be items $b_{11}, b_{12}, \dots, b_{1d}$. For each successive broadcast tick the adversary will generate d requests, a request for each item that was just broadcast. Specifically, requests $m + d(t-1) + 1, m + d(t-1) + 2, \dots, m + d(t-1) + d$ arrive at time t requesting items $b_{t1}, b_{t2}, \dots, b_{td}$ respectively, for $t = 1, 2, \dots, \left(\frac{n-m}{d}\right)$. This creates a schedule in which for each item there is exactly one request waiting at any time $t : 1 < t \leq \left(\frac{n-m}{d}\right)$. After time $t = \left(\frac{n-m}{d}\right)$ no more requests will arrive and $m - d$ requests will remain to be satisfied. These remaining requests will be satisfied in $\lceil \frac{m-d}{d} \rceil$ additional ticks. That is, the schedule will be completed at the end of time $t = \left(\frac{n-m}{d}\right) + \lceil \frac{m-d}{d} \rceil$. Let A denote the on-line algorithm. Considering the total wait time of the schedule, we notice that the total wait time of all requests for each item is $\frac{n-m}{d} + 1$ up until arrivals discontinue. The $m - d$ requests remaining to be satisfied will experience additional wait time over the required $\lceil \frac{m-d}{d} \rceil$ ticks needed to complete the schedule. We see that the total wait time for all remaining requests is $\sum_{l=1}^{\lceil \frac{m-d}{d} \rceil} (m - dl)$. Summing up we have:

$$\begin{aligned}
 \sum_{j=1}^n w_j(A) &= \sum_{j=1}^m \left(\frac{n-m}{d} + 1 \right) + \sum_{l=1}^{\lceil \frac{m-d}{d} \rceil} (m - dl) \\
 &= \frac{m(d+n-m)}{d} + \sum_{l=1}^{\lceil \frac{m-d}{d} \rceil} (m - dl) \\
 &= \frac{m(d+n-m)}{d} + (m-d) + (m-2d) + \dots + \left(m - \left\lceil \frac{m-d}{d} \right\rceil d \right) \\
 &= \frac{m(d+n-m)}{d} + m \left\lceil \frac{m-d}{d} \right\rceil - \frac{d \left\lceil \frac{m-d}{d} \right\rceil \left(\left\lceil \frac{m-d}{d} \right\rceil + 1 \right)}{2}
 \end{aligned}$$

We now argue that for the same input instance there exists an optimal schedule in which $w_j = 1$ for most requests. In the trivial case of $c \geq m$ the optimal schedule is equivalent to the on-line schedule. If $c < m$ then we must closely observe the input provided by the adversary. Because the adversary will always generate requests to replace requests that have just been satisfied by the on-line algorithm, we note that the on-line algorithm is never able to satisfy more than one request per item broadcast. In contrast, there exists a schedule constructed with knowledge of the future in which an item is selected to be broadcast at the tick in which most of the most immediate requests for that item have already arrived.

For instance, assume $c = 1$ and two requests j_1, j_2 arrive at time $t = 0$ for different items. Say also that we are given the knowledge that at time $t = 1$ another request j_3 will arrive such that the item requested by j_1 is the same item requested by j_3 . We see it is more efficient to broadcast item 2 at time $t = 1$ and item 1 at time $t = 2$ rather than vice versa as this minimizes the sum of the wait times experienced by all three requests j_1, j_2, j_3 .

It is with this motivation that the optimal schedule is constructed. An algorithm with complete knowledge of the input generated by the adversary will choose to satisfy a request for an item such that the difference between the current time and the arrival time of the next request for item that item is maximized. (The distance is assumed to be infinity if a request is never followed by a request for the same item.) In this way, requests generated by the adversary will tend to be for the same items as other requests already arrived thus decreasing the number of unique items requested.

Let OPT denote the algorithm by which the optimal schedule is created. Consider that at time $t = 1$ OPT knows that m unique requests have arrived at $t = 0$ and d additional requests will arrive at $t = 1$. Based on the above selection strategy OPT will choose d items

to broadcast at time $t = 1$. If any of these items correspond to those items to be requested at time $t = 1$, then necessarily it must be true that $m < 2d$. In this case it is obvious that all requests arriving before or at $t = 2$ will be satisfied at the end of $t = 2$, otherwise each item broadcast will not be requested at $t = 1$ and the number of unique requests at $t = 2$ will be $m - d$. That is, in contrast to the on-line schedule in which there was a request for every item throughout the beginning of the schedule, the optimal schedule has already eliminated up to d unique requests each tick. By continuing this method of broadcasting, the optimal schedule is able to quickly satisfy all the initial requests that have arrived at or before time $\lceil \frac{m}{d} \rceil$.

From time $\lceil \frac{m}{d} \rceil$ and beyond we know that the adversary generates at most d requests per tick. Thus all remaining requests in the schedule will be satisfied after one time unit of minimum waiting.

The total wait time of all requests in the schedule may be calculated in two parts: the amount of wait time of all requests before time $\lceil \frac{m}{d} \rceil$ and the amount of wait time of all requests after.

Before time $\lceil \frac{m}{d} \rceil$ we see that d requests must wait 1 time unit each before being satisfied at time $t = 1$. From time $t = 2$ to time $\lceil \frac{m}{d} \rceil$, each satisfied request will either have just arrived or will have been waiting since time $t = 0$ described by $\sum_{l=2}^{\lceil \frac{m}{d} \rceil} d(l + 1)$. Likewise, those requests satisfied at time $\lceil \frac{m}{d} \rceil$ will either have just arrived, for a total wait among those requests of $d \lceil \frac{m}{d} \rceil - m$, or will have been waiting since time $t = 0$ for a total wait of $(\lceil \frac{m}{d} \rceil + 1) (d - d \lceil \frac{m}{d} \rceil + m)$.

The d requests per tick arriving after time $\lceil \frac{m}{d} \rceil$ will be satisfied after one time unit of minimum waiting. Recalling there are $\lceil \frac{n-d}{d} \rceil$ ticks required to complete a schedule, this

results in $d \cdot (\lceil \frac{n-d}{d} \rceil - \lceil \frac{m}{d} \rceil)$ total wait time among these requests.

Considering all n requests we have:

$$\sum_{j=1}^n w_j(OPT) =$$

$$d \cdot \left(\left\lceil \frac{n-d-m}{d} \right\rceil \right) + d + \sum_{l=2}^{\lfloor \frac{m}{d} \rfloor} d(l+1) + d \left\lceil \frac{m}{d} \right\rceil - m + \left(\left\lceil \frac{m}{d} \right\rceil + 1 \right) \left(d - d \left\lceil \frac{m}{d} \right\rceil + m \right)$$

Therefore, for a fixed m, d and an arbitrarily large n , the ratio of $\sum_{j=1}^n w_j(A)$ over $\sum_{j=1}^n w_j(OPT)$ is a bound that approaches

$$\lim_{n \rightarrow \infty} \frac{\frac{m(d+n-m)}{d}}{d \left\lceil \frac{n-d-m}{d} \right\rceil} = \frac{m}{d}.$$

Since algorithm A is arbitrary throughout the proof, such a lower bound holds for all on-line algorithms for broadcast scheduling with c channels and unit-sized items. Note as well that this result indicates the competitive optimality of the earlier algorithms FCFS and MRF. ■

3.4 Simulation Results

The results in the previous section indicate that FCFS and MRF show the same worst-case behavior as measured by the competitive ratio. The relative frequency of such scenarios in practice, however, depends on the properties of the request sequences these algorithms actually face. In the average case, we do expect MRF to outperform FCFS in most real scenarios since MRF locally minimizes the objective function.

3.4.1 Simulation Parameters

In order to develop a feel for the behavior of each algorithm, we develop a discrete-event simulation to model the effects of various input sequence properties on the performance of FCFS and MRF. The parameters to the simulation are the properties of the input sequence that are of interest. For example, we may look at how each algorithm is affected as we increase the rate at which requests arrive or the number of requests arriving.

Each replication (simulated instance) consists of stochastically generating an arrival sequence, simulating the server's scheduling decisions according to the algorithm under study, and collecting data to compute the sum of wait times statistic. Because the terminal condition is specified by the number of processed requests, the terminal time (makespan) will vary from instance to instance. Large numbers of replications are used for each parameter change to ensure that the behavior observed is not dependent on any one run, i.e. are generated at 95% confidence.

In the absence of motivation for any other distribution, the arrival rate of requests is generated through use of a discrete version of the Exponential distribution. This rate measures the number of requests arriving per tick. Requests will stop arriving after the specified n requests have been generated. Each request is for an item drawn from a discrete version of the Uniform distribution. Thus, given that there are requests left to arrive, each tick will see the generation of a number of new requests that is, on average, equal to the arrival rate all of which will be for a specific item i with probability $\frac{1}{m}$.

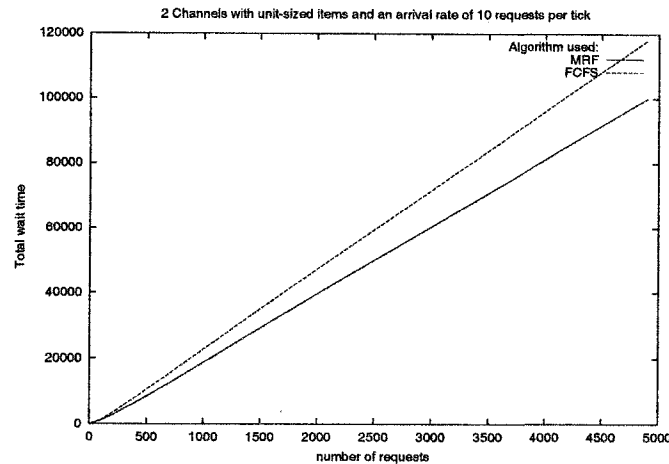
All the following results are obtained from simulation runs in which all items are unit-sized. Note that in terms of implementation the scheduling algorithm must be run for each

channel each system tick. Furthermore, because situations arise in which the scheduling algorithm generates a tie (more than one item is selected as the best item to send), these situations are resolved by running a second algorithm on the contending items. For instance, if four items appear to be equally good choices under FCFS then MRF will be used to determine which of those four to broadcast (the item with the largest number of requests). Should the matter still be unresolved after applying the second algorithm, priority is given to the smallest item index. Resolving ties is a non-trivial matter in terms of real-time algorithm complexity as is discussed later.

Due to the NP-Hardness of the problem in this model, note that the optimal schedule for each instance is never computed. The performance of each algorithm must therefore be studied based on a strict report of total wait time.

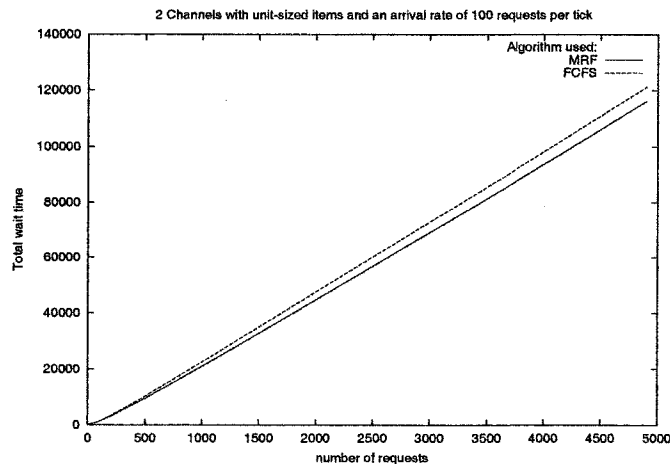
3.4.2 Performance Comparisons

Perhaps the most straightforward property of an input sequence is the sheer volume of requests it throws at a scheduling algorithm. For a given number of channels, a static number of distinct items, and an arrival rate it is obvious from Figures 3.4 and 3.5 that increasing the number of requests necessarily increases the sum of the wait times. Figures 3.4 and 3.5 compare, between the two algorithms, the effects of increasing the number of requests on the sum of wait times for a static arrival rate of 10 and 100 respectively. The number of data items is static at 100. Each point in the plot is derived from the mean of 1000 iterations and, as such 95% confidence intervals are too small to display. Note that as a tie breaker, each algorithm uses the other. That is, FCFS uses MRF to determine which request to satisfy in case of a tie and vice versa.

Figure 3.4: IMPACT OF INCREASING NUMBER OF REQUESTS ON TOTAL WAIT TIME, 10 REQUESTS PER TICK

In addition, we observe that though MRF outperforms FCFS as expected, the difference in performance decreases as the arrival rate increases. The reason for this trend is intuitive. MRF targets the most popular items when broadcasting and thus will generally satisfy more requests at low arrival rates than FCFS which will happily satisfy a single request that arrived earlier rather than 10 requests for the same item that arrived later. However, as the arrival rate increases to extremely large values, the number of requests for any item is so large that FCFS will satisfy a number of requests regardless of what item it selects, diminishing the advantage of MRF. This behavior is consistent with the analytical results described in the previous section.

In order to illustrate the dimensionality of parameter interaction, consider the point in the above figures where the number of requests has reached 200. If, at that point, we hold the number of requests constant at 200 and consider the effect on both figures of altering the arrival rate we generate Figure 3.6. Figure 3.6 compares, between the two algorithms

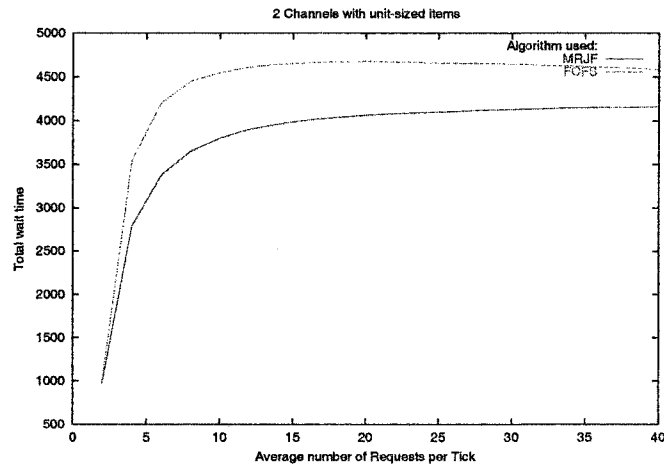
Figure 3.5: IMPACT OF INCREASING NUMBER OF REQUESTS ON TOTAL WAIT TIME, 100 REQUESTS PER TICK

FCFS and MRF, the effects of increasing request arrival rate on the sum of wait times for 200 requests, 2 channels, and 50 items. Each point in the plot is derived from the mean of 1000 iterations and, as such 95% confidence intervals are too small to display. Note that as a tie breaker, each algorithm uses the other, i.e. FCFS uses MRF to determine which request to satisfy in case of a tie.

We see from Figure 3.6 that increasing the arrival rate results in a narrowing of performance difference between the two algorithms. That is, as a system approaches saturation the advantages of MRF over FCFS diminish.

In Figure 3.6, it is assumed that the number of distinct items is constant as the arrival rate increases. Intuitively we also expect that increasing the number of distinct items will cause the sum of wait times to increase.

As the requests coming in are more distributed across a larger number of items, each item broadcast on a channel will satisfy fewer requests. In fact, the impact of increasing

Figure 3.6: IMPACT OF INCREASING ARRIVAL RATE ON TOTAL WAIT TIME

the number of distinct items is significantly larger than that of increasing the arrival rate. Asymptotically, increasing the arrival rate does nothing more than add requests for items that already needed to be sent. That is, assuming no more arrivals, the difference in 100 requests for all items and 200 requests for all items is irrelevant in that m broadcasts will complete either schedule and the wait time experienced by each requests is not remarkable. However, the difference between 1 request for each of 100 items as opposed to 200 items is extreme in that the last item satisfied has experienced a wait time an order of magnitude longer than the first request satisfied. The plots in Figures 3.7 show these trends. For any given arrival rate, the increase in the number of data items is seen to increase the sum of wait times noticeable. In contrast, for a given number of items, an increase in the arrival rate is less apparent. We observe that the effect of increasing the number of items is more pronounced for MRF than for FCFS as is to be expected. This is another indication that the number of distinct items m is a significant factor in the analytical performance between

the two algorithms.

As stated earlier in this section, each scheduling algorithm can encounter situations in which multiple items are equal candidates to be selected for broadcasting. This occurs when an algorithm produces more candidates than the number of channels. That is, if the FCFS algorithm finds 6 items with requests that arrived at the same earliest time and there are 6 channels then all items will be broadcast. However, if 7 candidates are found then a decision must be made as to which 6 of the 7 will be sent requiring another algorithm to be used as a “tie breaker”. The running of this second algorithm will slow the real-time performance of the on-line algorithm (and impact the simulation complexity).

The following plots indicate that a tie breaking algorithm must be used increasingly as the arrival rate increases and as the number of distinct items increases. We notice that when the number of data items is sufficiently large, a tie breaking algorithm is consulted with virtually every decision. From the perspective of algorithm design, these results indicate that the issue of how a tie will be resolved computationally is an area that can greatly affect the performance of these two algorithms in particular.

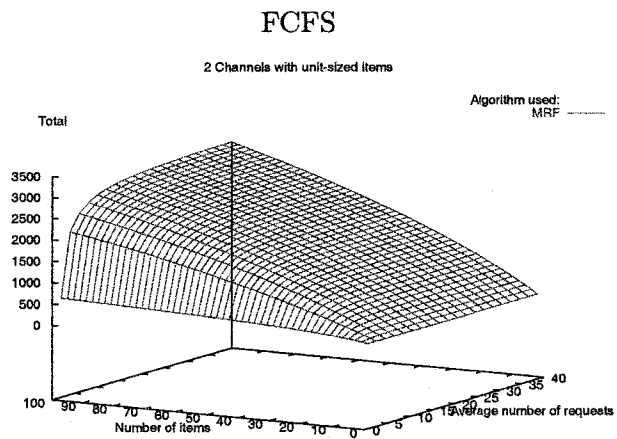
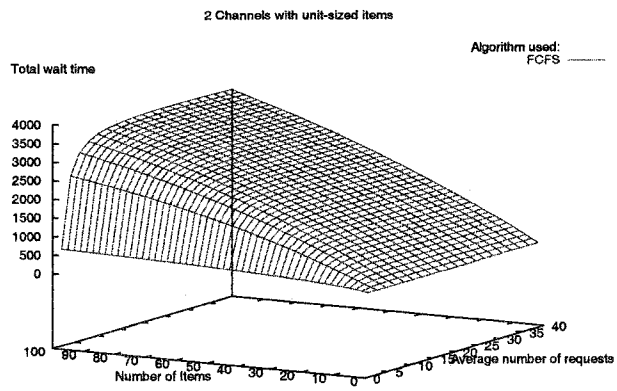
The plots in Figure 3.8 measure the effects of increasing request arrival rate and increasing number of items on the sum of percentage of decisions that must use a tie-breaking algorithm. For instance, if there are 2 channels and the original algorithm produces 4 items of equal importance, the tie-breaking algorithm must be used twice, once for each decision. Note that 2 items of equal importance for 2 channels is no tie at all. As a tie breaker, each algorithm uses the other. FCFS uses MRF to determine which request to broadcast in case of a tie. If the second algorithm produces yet more ties, the item to be broadcast is determined by smallest data item index first. Each vertex in the plot is a point derived

from the mean of 1000 iterations and, as such 95% confidence intervals are too small to display.

3.4.3 Simulation Conclusions

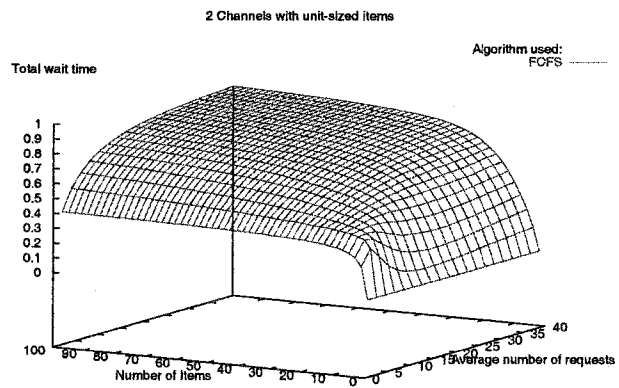
Though FCFS and MRF have equal competitive ratios, simulation results indicate that MRF outperforms FCFS on average. The performance gain of MRF over FCFS diminishes, however, as the number of requests, arrival rate, and number of data items increases. These observations agree with performance measures found in the literature and complement our analytical results. The NP-Hardness of the problem combined with the complexity burden of tie breaking, even with naive algorithms, establishes the simulation of this model as computationally intensive.

Figure 3.7: ALGORITHM PARAMETERS IN 3 DIMENSIONS - MRF

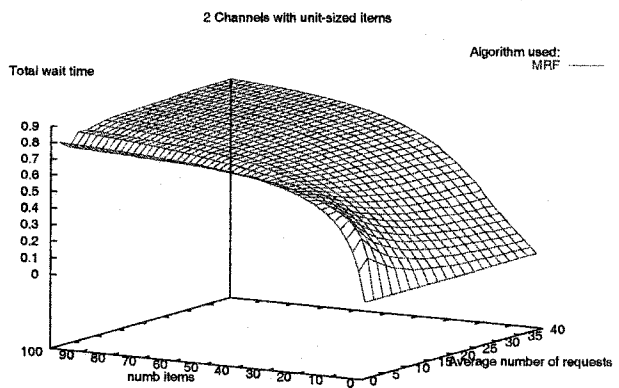


MRF

Figure 3.8: THE COMPUTATIONAL BURDEN OF TIE RESOLUTION



FCFS



MRF

Chapter 4

Minimizing Two Metrics: The Single-Item Case

We now consider the Data Broadcast Scheduling Problem in which the server attempts to maximize the quality of service as perceived by both clients and server. Client quality of service depends upon the speed at which their requests are satisfied and server quality of service is based upon the cost of satisfying those requests. Thus, the server will attempt to minimize two metrics: total wait time and total broadcast cost. As such, the server's two goals are said to be conflicting in that increasing the number of broadcasts will tend to reduce client wait time as it increases server cost and vice versa. In this chapter we present both analytical and simulation results regarding algorithms that attempt to optimize both metrics simultaneously. While the introduction of broadcast cost has been applied to push-based broadcast scheduling [33, 34, 43], an original contribution of our work is to apply this metric to the pull-based model. This chapter first formally defines the model and then proceeds to describe the analytical and simulation results of two online algorithms obtained within its context.

4.1 Model Description

Creating a special case of the general problem definition established in Section 1.3, we are given a server with access to a single unit-sized data item and a single channel over which the item may be transmitted. A sequence of requests is also given where each request arrives for that item. Due to the presence of a single item only, such an input sequence can be succinctly described as a sequence of pairs where each pair describes the arrival time a_j of a group of requests. That is, $(a_1, n_1), (a_2, n_2), \dots, (a_k, n_k)$ describes an input sequence where a_j is the arrival time of the j th group of requests and n_k is the number of requests that arrived at that time. Note that $n_1 + n_2 + \dots + n_k = n$, where k is the total number of groups arriving and n is the total number of requests. At the beginning of each system tick, the server must decide whether or not to broadcast the item. That is, the server may refrain from broadcasting the item (referred to as “waiting”) even when requests are in queue. This is in contrast to models, such as the one described in Chapter 3, in which the server attempts only the minimization of total wait time. Unlike those models, it makes sense for a server with a broadcast cost metric to sometimes wait even with requests in queue when the cost of waiting is smaller than the cost of broadcasting. In order to maximize the quality of service this system provides to its clients and itself, the server will attempt to minimize the total cost of a schedule S determined by the objective function

$$OBJ(S) = \sum_{j=1}^n w_j + B \cdot \beta,$$

where the wait time of each request j , w_j , is summed over the total number of requests n and the total number of broadcasts B in schedule S is multiplied by the cost of single

broadcasting β^1 .

4.2 Algorithms

In this chapter we consider two online algorithms the server may employ to minimize the above objective function, a naive algorithm we will call LAZY and a more sophisticated algorithm we will call GREEDY. We also present a polynomial-time offline algorithm based upon the dynamic programming technique that is shown to be optimal.

The online algorithm LAZY always broadcasts immediately after the arrival of a request group. Its name derives from the fact that it never considers other factors affecting the quality of a schedule, such as the broadcast cost.

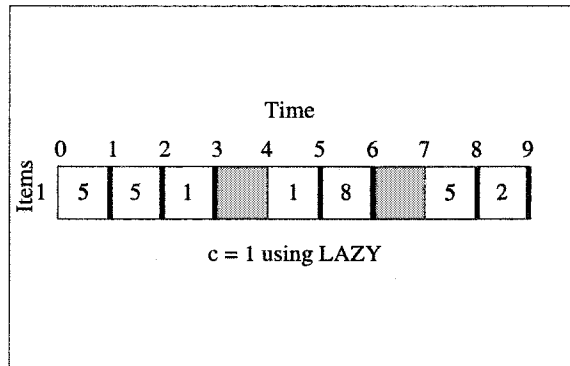
As an example, consider a $\beta = 5$ and an input sequence $(0, 5), (1, 5), (2, 1), (4, 1), (5, 8), (7, 5), (8, 2)$. LAZY constructs the schedule in Figure 4.1 with a total wait time of 27, a total broadcast cost of 35, and a total schedule cost of 62.

Designing a smarter online approach than that above, the GREEDY algorithm attempts to make a good broadcasting decision by considering both the broadcast cost β and the amount of accumulated wait time of all unsatisfied requests. That is, it compares the amount of wait time unsatisfied requests have accumulated while in queue to the cost of satisfying them. If β is large in comparison to this total accumulated wait time, the algorithm may choose to wait until such a time as the cost of broadcasting is more warranted.

More specifically, let t be the time at which GREEDY must make a decision to wait or broadcast at the next time $t + 1$. Let P be the number of pending (unsatisfied) requests

¹Because broadcast cost and wait time are measured in different units, some weighted measure between the two is desired. Thus the objective function might be written as $\alpha(\sum_{j=1}^n w_j) + B \cdot \beta'$. However, for simplicity β' can be normalized to β , the relative cost of broadcasting relative to a unit of wait time.

Figure 4.1: EXAMPLE SCHEDULING DIAGRAM OF LAZY



that have arrived up to time t , inclusively, since the last broadcast was made and let Q be the accumulated wait time of all P pending requests at time $t + 1$. The pseudocode for GREEDY is given below.

GREEDY Online Algorithm

Single broadcast cost: β

Input request sequence: $(a[1], n[1]), \dots, (a[k], n[k])$

$t = a[1]$

$P = n[1]$

$Q = n[1]$

do

 if $\beta \leq Q$

 then broadcast at $t+1$

$Q = 0$

$P = 0$

 else wait at $t+1$

$Q = Q + P$

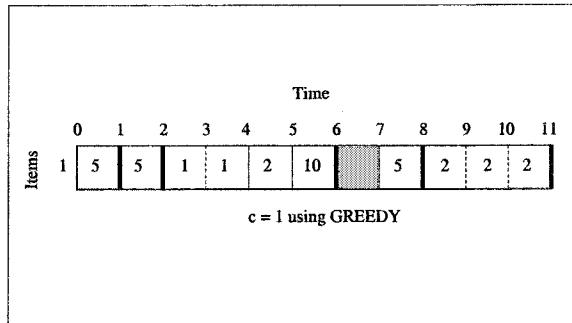
$t = t + 1$

 if $t == a[i]$ for some i

 then $Q = Q + n[i]$

$P = P + n[i]$

Until all requests have arrived and have been satisfied

Figure 4.2: EXAMPLE SCHEDULING DIAGRAM OF GREEDY

Using the same input examined with the LAZY algorithm above with $\beta = 5$ and the input sequence $(0, 5), (1, 5), (2, 1), (4, 1), (5, 8), (7, 5), (8, 2)$, GREEDY constructs the schedule in Figure 4.2 with a total wait time of 35, a total broadcast cost of 25, and a total schedule cost of 60.

Though more sophisticated than the LAZY algorithm, GREEDY does not always provide an optimal solution. An optimal solution can always be found, however, through use of an offline dynamic programming algorithm. Given a broadcast cost β and an input sequence of $(a_1, n_1), (a_2, n_2), \dots, (a_k, n_k)$, consider that a schedule satisfying all requests in this sequence is making a binary decision to wait or broadcast at each opportunity. Note as well, that a broadcasting opportunity appears only after the arrival of requests. Without arrivals in a previous tick, a server that has decided to wait should continue to do so (else the decision to wait was in error). In a sense, the dynamic programming algorithm checks all possible ways a server could satisfy the sequence of requests and chooses the schedule that minimizes the objective function. The construction of this algorithm follows.

Via the methodology of dynamic programming, we begin with the definition of a function

that returns the optimal solution for a partial input sequence. Let $C(i, j)$ be the minimum cost (total wait time plus total broadcast cost) of all schedules for the request sequence $(a_i, n_i), \dots, (a_j, n_j)$, for $1 \leq i \leq j \leq k$. (Here we use the indices's i, j, k, l apart from their previous associations.) Thus, $C(1, k)$ provides the minimum cost for the original request sequence $(a_1, n_1), \dots, (a_k, n_k)$. Function $C(i, j)$ can be defined recursively as follows:

$$C(i, j) = \begin{cases} n_i + \beta & \text{if } i = j \\ \min\{C_1(i, j), \min_{i \leq l < j} \{C(i, l) + C(l + 1, j)\}\} & \text{if } i < j \end{cases}$$

where $C_1(i, j) = \sum_{l=i}^j n_l(a_j + 1 - a_l) + \beta$, which is the cost of the schedule that broadcasts only once at $a_j + 1$ to satisfy all requests in $(a_i, n_i), \dots, (a_j, n_j)$. That is, C_1 establishes the cost of broadcasting a single time at the end of a given interval. Note that the computation of $C(i, j)$ for $i < j$ defined above considers two possibilities: one is to broadcast only once incurring the cost of $C_1(i, j)$ and the other is to broadcast at least twice, at time $a_l + 1$ and at time $a_j + 1$, with l to be determined by calculating the minimum incurred cost $\min_{i \leq l < j} \{C(i, l) + C(l + 1, j)\}$.

The pseudocode for our algorithm is given below. Two $k \times k$ tables are used in the algorithm: table $C1[i, j]$ to store $C_1(i, j)$ and table $C[i, j]$ to store $C(i, j)$.

Optimal Offline Algorithm

Single broadcast cost: beta

Input request sequence: (a[1],n[1]), ..., (a[k],n[k])

//Construction of the C1 table by the definition of the C1 function

for i from 1 to k

 for j from i to k

 C1[i, j] = beta

 for l from i to j

Table 4.1: EXAMPLE CONSTRUCTION of C1

i,j	1	2	3	4	5	6	7
1	10	20	31	54	74	119	146
2	*	10	16	29	44	79	101
3	*	*	6	9	19	44	61
4	*	*	*	6	15	38	54
5	*	*	*	*	13	34	49
6	*	*	*	*	*	10	17
7	*	*	*	*	*	*	7

```

      C1[i,j] = C1[i,j] + n[l] * (a[j]+1-a[l])
//Construction of the C table by the dynamic programming method
for i from 1 to k
  C[i,i] = C1[i,i]
for j from 2 to k
  for i from j-1 to 1
    temp_min = +infinity
    for l from i to j
      temp_min = min {temp_min, C[i,l] + C[l+1,j]}
    C[i,j] = min {C1[i,j], temp_min}
return C[1,k]

```

The time complexity of the above algorithm is $O(k^3)$. This algorithm only returns the cost of the optimal broadcast schedule. As shown later in this section, however, it can be modified to give the actual broadcast schedule without heavily impacting this complexity.

Consider the same input sequence used in the examples of Lazy and Greedy. Given $\beta = 5$ and an input sequence $(0, 5), (1, 5), (2, 1), (4, 1), (5, 8), (7, 5), (8, 2)$, the above algorithm produces Tables 4.1 and 4.2.

From entry $C[1, 7]$ it can be seen that the algorithm produces a schedule with a total cost of 56. To actually determine that schedule, we may modify the original algorithm to track the points at which broadcasts are performed. As table C is constructed, a comparison is

Table 4.2: EXAMPLE CONSTRUCTION OF C

i,j	1	2	3	4	5	6	7
1	10	20	26	29	39	49	56
2	*	10	16	19	29	39	46
3	*	*	6	9	19	29	36
4	*	*	*	6	15	25	32
5	*	*	*	*	13	23	30
6	*	*	*	*	*	10	17
7	*	*	*	*	*	*	7

made for each interval (each table entry) to determine if the schedule cost will be minimized by single or multiple broadcasts for that interval. By tracking the decision made at each such comparison it is possible to reconstruct the optimal schedule once the final cost has been calculated. This process is enabled by the construction of a third table T done in tandem with the construction of table C .

Table T is constructed as follows. At any point in the construction of table C at which the optimal decision is determined to be a single broadcast within an interval, we record a -1 in the corresponding entry of table T at cell $T[i, j]$. If the optimal decision is instead determined to occur in at least two locations within the interval, we record the l value that marks the two intervals. The modification to the algorithm pseudocode for constructing the T table is given below.

```

Optimal Offline Algorithm
Single broadcast cost: beta
Input request sequence: (a[1],n[1]), ..., (a[k],n[k])

//Construction of the C1 table by the definition of the C1 function
for i from 1 to k
  for j from i to k
    C1[i,j] = beta

```

Table 4.3: EXAMPLE CONSTRUCTION OF T

i,j	1	2	3	4	5	6	7
1	-1	-1	0	0	0	0	0
2	*	-1	-1	1	1	1	1
3	*	*	-1	-1	-1	4	4
4	*	*	*	-1	-1	4	4
5	*	*	*	*	-1	4	4
6	*	*	*	*	*	-1	-1
7	*	*	*	*	*	*	-1

```

    for l from i to j
        C[i,j] = C1[i,j] + n[l] * (a[j]+1-a[l])
//Construction of the C and T tables by the dynamic programming method
for i from 1 to k
    C[i,i] = C1[i,i]
    T[i,i] = -1
for j from 2 to k
    currentL = 0
    for i from j-1 to 1
        temp_min = +infinity
        for l from i to j
            if temp_min > C[i][l] + C[l+1][j]
                temp_min = C[i][l] + C[l+1][j];
                currentL = l;
        if C1[i][j] > temp_min
            C[i][j] = temp_min
            T[i][j] = currentL
        else
            C[i][j] = C1[i][j]
            T[i][j] = -1
return C[1,k]

```

For the given example input instance, Table 4.3 is generated.

Once all three tables have been constructed, a recursive construction of the schedule is performed. Starting with the last decision made, table entry $T[1, k]$, the value of each entry is examined. If the entry contains a -1 , then a broadcast has occurred at the end of that interval. If the entry is not -1 , then it is an l value and a call is made recursively to

determine the decisions made by the two intervals ending and beginning at decision point l respectively. In pseudocode this procedure is below.

```

ReconstructSchedule(T, k, a, b)

l = T[a,b]
if l == -1
    broadcast at b
else
    ReconstructSchedule(T, k, a, l)
    ReconstructSchedule(T, k, l+1, b)

```

For the T table established for this example, the procedure executes as follows (with the nested structure representing recursive calls).

```

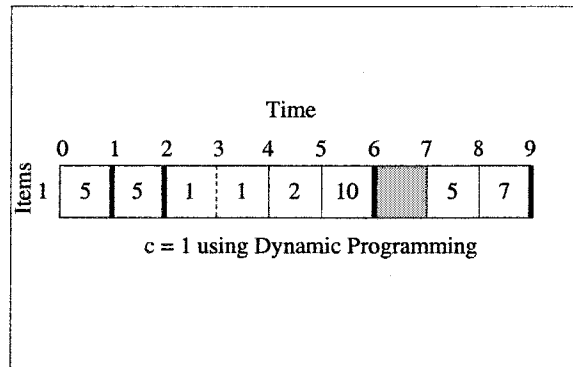
ReconstructSchedule(T, 7, 1, 7) 'l = 1'
  ReconstructSchedule(T, 7, 1, 1) 'l = -1 -> BROADCAST AT 1'
  ReconstructSchedule(T, 7, 2, 7) 'l = 2'
    ReconstructSchedule(T, 7, 2, 2) 'l = -1 -> BROADCAST AT 2'
    ReconstructSchedule(T, 7, 3, 7) 'l = 5'
      ReconstructSchedule(T, 7, 3, 5) 'l = -1 -< BROADCAST AT 5'
    ReconstructSchedule(T, 7, 6, 7) 'l = -1 -> BROADCAST AT 7'

```

This provides the schedule in Figure 4.3 with a total wait time of 36, a total broadcast cost of 20, and a total schedule cost of 56.

4.3 Analytical Results

As stated earlier, the wait time experienced by any request j for a unit-sized item is $w_j \geq 1$. Thus the minimum total wait time of any schedule independent of the scheduling algorithm is at least n . That is, $\sum_{j=1}^n w_j \geq \sum_{j=1}^n 1 = n$. The minimum broadcast cost for any schedule is β since, at the very least, a broadcast is required at the end of a schedule.

Figure 4.3: EXAMPLE SCHEDULING DIAGRAM OF DYNAMIC PROGRAMMING ALGORITHM

Therefore, $n + \beta$ is a lower bound to the total cost for any schedule, including the optimal schedule. While $n + \beta$ is a lower bound, it is not a tight bound. The following subsections address the need for a tight bound for the LAZY and GREEDY algorithms respectively.

4.3.1 Competitive Ratio for the LAZY Algorithm

LAZY broadcasts its item at any opportunity at which a request can be satisfied. Though there are n requests in the input sequence, some of these requests may arrive at the same time. As such, let k be the number of distinct arrival times (also the number of request groups that arrive) such that $k \leq n$. Thus LAZY will make exactly k broadcasts and each request will experience unit wait time. Clearly then, for any instance $(a_1, n_1), \dots, (a_k, n_k)$ with k arrival times, $n = \sum_{j=1}^k n_j \geq k$ requests, and the single broadcast cost of β , the cost of the schedule constructed by algorithm LAZY is

$$C = n + k\beta.$$

Here we define C to be the total wait time plus total broadcast cost of the schedule

Table 4.4: INSTANCES FOR LAZY

Instance	Optimal cost
$I: (a_1, n_1), (a_2, n_2), \dots, (a_k, n_k)$	C^*
$I_1: (1, n_1), (2, n_2), \dots, (k, n_k)$	C_1^*
$I_2: (1, 1), (2, 1), \dots, (k, n - k + 1)$	C_2^*
$I_0: (1, 1), (2, 1), \dots, (k, 1)$	C_0^*

given by LAZY for any instance. Likewise, C^* is the total wait time plus total broadcast cost of the optimal solution for the same instance. We will show that for any instance, $C/C^* \leq (\beta + 1)/(\sqrt{2\beta} + 1/2)$. However, in order to establish the competitive ratio of LAZY, it is necessary to obtain a tighter lower bound on the cost of the optimal schedule for any instance with k arrival time and n requests.

Lemma 4.1 *Let C^* be the cost of the optimal schedule for any instance with k arrival times and $n = \sum_{j=1}^k n_j$ requests. Let C_0^* be the cost of the optimal schedule for the instance with $a_j = j$ and $n_j = 1$ for $j = 1, \dots, k$. Then $C^* \geq C_0^* + n - k$.*

Proof: Given any instance $I: (a_1, n_1), \dots, (a_k, n_k)$ with $n = \sum_{j=1}^k n_j$, we define the following instance variations:

We prove the lemma in three steps. In the first step we show by a series of transformations applied to the schedules that $C^* \geq C_1^*$. In the second step we show by a series of transformations applied to the schedules that $C_1^* \geq C_2^*$. In the third step we show that $C_2^* = C_0^* + n - k$ by observing that the optimal broadcast schedules are the same for instances I_2 and I_0 , shown in Table 4.4 while the optimal costs have a difference of $n - k$. Combining the three inequalities, we have our lemma. ■

Now consider C_0^* , the cost of the optimal schedule for request sequence $I_0 = (1, 1), (2, 1), \dots, (k, 1)$ with k arrival groups and unit inter-arrival times. The following lemma gives a lower bound on C_0^* .

Lemma 4.2 *Let C_0^* be the cost of the optimal schedule for request sequence $I_0 = (1, 1), (2, 1), \dots, (k, 1)$. Let β be the single broadcast cost. Then $C_0^* \geq k(\sqrt{2\beta} + 1/2)$.*

Proof: In instance $I_0 = (1, 1), (2, 1), \dots, (k, 1)$, the number of requests n is equal to k , the number of distinct arrival times. Let l be the number of broadcast events in a schedule for I_0 . If l is fixed, then the optimal schedule with exactly l broadcast events is to broadcast at time instants that spread out as evenly as possible in the time interval $[2, k + 1]$. More specifically, the optimal schedule which broadcasts l times will see that each of the first $l - k \bmod l$ broadcast events satisfies $\lfloor k/l \rfloor$ requests, accumulating $(1/2)\lfloor k/l \rfloor(\lfloor k/l \rfloor + 1)$ in wait time. Each of the last $k \bmod l$ broadcast events satisfies $\lceil k/l \rceil$ requests, accumulating $(1/2)\lceil k/l \rceil(\lceil k/l \rceil + 1)$ in wait time.

Define function $cost(l)$ to be the cost (total wait time plus total broadcast cost) of the optimal schedule among all schedules which broadcast exactly l times. Then we have

$$\begin{aligned} cost(l) &= (l - k \bmod l) \cdot (1/2)\lfloor k/l \rfloor(\lfloor k/l \rfloor + 1) + (k \bmod l) \cdot (1/2)\lceil k/l \rceil(\lceil k/l \rceil + 1) + l\beta \\ &= l \cdot (1/2)\lfloor k/l \rfloor(\lfloor k/l \rfloor + 1) + l\beta + (k \bmod l)\lceil k/l \rceil. \end{aligned} \quad (4.1)$$

Thus the cost of the optimal schedule, considering all possible number of broadcast events, is

$$C_0^* = \min_{1 \leq l \leq k} \{cost(l)\} = \min \left\{ \min_{1 \leq l \leq k, k \bmod l = 0} \{cost(l)\}, \min_{1 \leq l \leq k, k \bmod l \neq 0} \{cost(l)\} \right\}. \quad (4.2)$$

Consider two cases of l . First assume $k \bmod l = 0$. Then

$$\text{cost}(l) = l \cdot (1/2)(k/l)(k/l + 1) + l\beta.$$

Consider the corresponding continuous function $f(x) = x \cdot (1/2)(k/x)(k/x + 1) + x\beta$. By computing the derivative of $f(x)$, we find that the minimum of $f(x)$ occurs at $x = k/\sqrt{2\beta}$ and is $k(\sqrt{2\beta} + 1/2)$. So

$$\min_{1 \leq l \leq k, k \bmod l = 0} \{\text{cost}(l)\} \geq \min_{\forall x} \{f(x)\} = k(\sqrt{2\beta} + 1/2). \quad (4.3)$$

Next assume $k \bmod l \neq 0$. Let $k/l = \lfloor k/l \rfloor + \epsilon$ for some $\epsilon \in (0, 1)$. Then

$$\begin{aligned} \text{cost}(l) &= l \cdot (1/2)\lfloor k/l \rfloor(\lfloor k/l \rfloor + 1) + l\beta + (k \bmod l)\lceil k/l \rceil \\ &= l \cdot (1/2)\lfloor k/l \rfloor\lceil k/l \rceil + l\beta + (k \bmod l)\lceil k/l \rceil \\ &= \lceil k/l \rceil((l/2)\lfloor k/l \rfloor + k \bmod l) + l\beta \\ &= \lceil k/l \rceil(k - (l/2)\lfloor k/l \rfloor) + l\beta \\ &= (k/l + 1 - \epsilon)(k - (l/2)(k/l - \epsilon)) + l\beta \\ &= (k^2/2)(1/l) + k/2 + (1/2)l\epsilon(1 - \epsilon) + l\beta. \end{aligned}$$

Consider the corresponding continuous function $g(x) = (k^2/2)(1/x) + k/2 + (1/2)x\epsilon(1 - \epsilon) + x\beta$. By computing the derivative of $g(x)$, we find that the minimum of $g(x)$ occurs at $x = k/\sqrt{2\beta + \epsilon(1 - \epsilon)}$ and is $k(\sqrt{2\beta + \epsilon(1 - \epsilon)} + 1/2)$. So

$$\min_{1 \leq l \leq k, k \bmod l \neq 0} \{\text{cost}(l)\} \geq \min_{\forall x} \{g(x)\} = k(\sqrt{2\beta + \epsilon(1 - \epsilon)} + 1/2). \quad (4.4)$$

Combining the two cases, we have

$$\begin{aligned}
C_0^* &= \min\left\{\min_{1 \leq l \leq k, k \bmod l = 0} \{cost(l)\}, \min_{1 \leq l \leq k, k \bmod l \neq 0} \{cost(l)\}\right\} \quad \text{by Equation (4.2)} \\
&\geq \min\{k(\sqrt{2\beta} + 1/2), k(\sqrt{2\beta} + \epsilon(1 - \epsilon) + 1/2)\} \quad \text{by Equations (4.3) and (4.4)} \\
&= k(\sqrt{2\beta} + 1/2).
\end{aligned}$$

■

Now, we are ready to prove the competitive ratio for algorithm LAZY.

Theorem 4.1 *For any instance, $C/C^* \leq (\beta + 1)/(\sqrt{2\beta} + 1/2)$, where C and C^* are the costs of the LAZY schedule and of the optimal schedule (for the same instance), respectively.*

Proof: Recall that for any instance $C = n + k\beta$ and that by Lemmas 4.1 and 4.2 $C^* \geq C^* + n - k \geq k(\sqrt{2\beta} + 1/2) + n - k = n + k(\sqrt{2\beta} - 1/2)$. Therefore,

$$\begin{aligned}
\frac{C}{C^*} &\leq \frac{n + k\beta}{n + k(\sqrt{2\beta} - 1/2)} \\
&= 1 + \frac{\beta - \sqrt{2\beta} + 1/2}{n/k + \sqrt{2\beta} - 1/2} \\
&\leq 1 + \frac{\beta - \sqrt{2\beta} + 1/2}{1 + \sqrt{2\beta} - 1/2} \quad \text{since } n/k \geq 1 \\
&= \frac{\beta + 1}{\sqrt{2\beta} + 1/2}.
\end{aligned}$$

■

Theorem 4.2 *There is an instance for which $C/C^* \geq (\beta + 1)/(\sqrt{2\beta} + 1/2)$, where C and C^* are the costs of the LAZY schedule and of the optimal schedule (for the same instance), respectively.*

Proof: Consider an instance $(1, 1), (2, 1), \dots, (k, 1)$, where $k = 2xy$ for some integers x and y . Note that $n = \sum_{j=1}^k n_j = k$. Let $\beta = 2x^2$. So

$$C = n + k\beta = k(1 + \beta) = 2xy(1 + \beta). \quad (4.5)$$

According to Equations (4.2) and (4.1) in the proof of Lemma 4.2, we have $C^* = \min_{1 \leq l \leq k} \{cost(l)\}$, where $cost(l) = l \cdot (1/2)[k/l]([k/l] + 1) + l\beta + (k \bmod l)[k/l]$. Choose $l' = k/\sqrt{2\beta} = (2xy)/(2x) = y$. We then have $k \bmod l' = k \bmod y = 2xy \bmod y = 0$. Therefore, $cost(l') = y \cdot (1/2) \cdot 2x(2x + 1) + y \cdot 2x^2 = yx(4x + 1)$. So,

$$C^* = \min_{1 \leq l \leq k} \{cost(l)\} \leq cost(l') = yx(4x + 1). \quad (4.6)$$

Considering the ratio C/C^* , we have

$$\begin{aligned} \frac{C}{C^*} &\geq \frac{2xy(1 + \beta)}{yx(4x + 1)} \quad \text{by Equations (4.5) and (4.6)} \\ &= \frac{2(1 + \beta)}{4 \cdot \sqrt{\beta/2} + 1} \quad \text{since } \beta = 2x^2 \\ &= \frac{\beta + 1}{\sqrt{2\beta} + 1/2}. \end{aligned}$$

■

The competitive ratio of LAZY is roughly proportional to $\sqrt{\beta}$. However, the ratio remains small when β is reasonably small. In particular, the ratio is bounded by 2 if $\beta \leq 8$, in which case the single broadcast cost weighs no more than eight times of one unit of wait time.

4.3.2 Competitive Ratio for the GREEDY Algorithm

Next we show that the competitive ratio of the GREEDY algorithm is 2, by proving that 2 is both the lower bound and the upper bound on the competitive ratio. In what follows, we call a schedule produced by algorithm GREEDY a GREEDY schedule.

Theorem 4.3 *There is an instance for which $C/C^* \geq 2$, where C and C^* are the costs of the GREEDY schedule and of the optimal schedule (for the same instance), respectively.*

Proof: Consider the request sequence with only one group of requests arriving at time 1: $(1, n_1)$. Assume that $\beta = l \cdot n_1$ for some large l . Clearly, the optimal schedule for the instance will broadcast at time 2. So

$$C^* = n_1 + \beta.$$

Now we apply algorithm GREEDY to the instance. Observe that at time $t = 1, \dots, l - 1$, the accumulated wait time A is $tn_1 < \beta$ and that at time $t = l$, the accumulated wait time A is $l \cdot n_1 \geq \beta$. So the algorithm will choose to wait at time $t = 1, \dots, l - 1$ until time l , when the algorithm will choose to broadcast to satisfy the requests. So

$$C = l \cdot n_1 + \beta.$$

Considering the ratio between C and C^* , we have

$$\frac{C}{C^*} = \frac{l \cdot n_1 + \beta}{n_1 + \beta} = \frac{2l}{l+1} = 2 - \frac{2}{l+1} \rightarrow 2$$

as $l \rightarrow \infty$. ■

Theorem 4.4 *For any instance, $C/C^* \leq 2$, where C and C^* are the costs of the GREEDY schedule and of the optimal schedule (for the same instance), respectively.*

Proof: For any instance $(a_1, n_1), \dots, (a_k, n_k)$, the GREEDY schedule can be partitioned into time intervals defined by broadcast times. To be specific, assume that in the GREEDY schedule the event of broadcast happens a total of l times, at times b_1, \dots, b_l . Let $b_0 = a_1$. Then the schedule can be partitioned into intervals $[b_0, b_1), [b_1, b_2), \dots, [b_{l-1}, b_l)$. For the i th interval $[b_{i-1}, b_i)$, define K_i to be the set of requests arriving in the interval, R_i to be the number of requests in K_i , and A_i to be the accumulated wait time by requests in K_i at time b_i . From the definition of algorithm GREEDY, we have

$$A_i - R_i < \beta \leq A_i. \quad (4.7)$$

Summing up the above inequalities for all $i = 1, \dots, l$, we get

$$\sum_{i=1}^l A_i - \sum_{i=1}^l R_i < l\beta \leq \sum_{i=1}^l A_i.$$

So

$$C = \sum_{i=1}^l A_i + l\beta \leq 2 \sum_{i=1}^l A_i.$$

To prove that $C/C^* \leq 2$, it suffices to prove that $C^* \geq \sum_{i=1}^l A_i$.

For any instance, we align its optimal schedule with its GREEDY schedule. As a result, the optimal schedule is also partitioned into intervals by the broadcast times in the GREEDY schedule. For the i th interval $(b_{i-1}, b_i]$ ² in the optimal schedule, let C^* be the

²Note that the time intervals used for partitioning the GREEDY schedule are $[)$ but the time intervals used for partitioning the optimal schedule are $(]$. This is not a typo.

cost incurred in the interval, which is the broadcast cost incurred in $(b_{i-1}, b_i]$ plus the wait time accumulated in $(b_{i-1}, b_i]$. We consider two cases for the interval.

Case 1. No broadcast occurs in $(b_{i-1}, b_i]$ in the optimal schedule. Then $C_i^* = A_i + \Delta \geq A_i$, where $\Delta \geq 0$ is nonzero only when there is not broadcast at b_{i-1} and there are requests arriving earlier than b_{i-1} but at or after the last broadcast time before b_{i-1} in the optimal schedule.

Case 2. There is at least one broadcast event in $(b_{i-1}, b_i]$ in the optimal schedule. Then $C_i^* \geq R_i + \beta$ since every request arriving in $[b_{i-1}, b_i)$ will accumulate at least one unit of wait time in $(b_{i-1}, b_i]$ and the broadcast cost incurred in the interval is at least β . Recall that $A_i - R_i < \beta$ in Equation (4.7). So $C_i^* \geq R_i + \beta > R_i + A_i - R_i = A_i$. Summing the inequality $C_i^* \geq A_i$ for all intervals, we get

$$C^* = \sum_{i=1}^l C_i^* \geq \sum_{i=1}^l A_i.$$

This proves the theorem. ■

4.4 Simulation Results

In order to develop a feel for the empirical behavior of each algorithm, we utilize a discrete-event simulation to model the effects of various input sequence properties on the performance of LAZY and GREEDY. Unlike models (such as that of Chapter 3) in which finding an optimal solution is NP-Hard, the model in this chapter is computationally easier to simulate due to the existence of the optimal offline algorithm developed in Section 4.2. The availability of this optimal algorithm allows a more direct comparison of online algorithm performance;

since the optimal schedule for any instance can be computed, we can determine the extent to which a schedule produced GREEDY or LAZY deviates from the optimal.

For the results to follow, each replication (simulated instance) consists of stochastically generating an arrival sequence, simulating the server's scheduling decisions according to the algorithm under study, and collecting data to compute the sum of wait times, sum of broadcast cost, and total schedule cost statistics. Because the terminal condition is specified by the number of processed requests, the terminal time (makespan) will vary from instance to instance. Large numbers of replications (1000 replications for results displayed in figures) are used for each parameter change to ensure that the behavior observed is not dependent on any one run. Thus, replications are generated such that statistics are reported at 95% confidence.

4.4.1 Simulation Parameters

Several parameters to the problem will affect the characteristics of the input sequence that is simulated on each algorithm for each replication: the arrival rate, the inter-arrival time, and the total number of requests set to arrive. In the absence of motivation for any other distribution, the arrival rate of requests is generated through use of a discrete version of the Exponential distribution. This rate measures the number of requests, or the size of the group of requests, arriving per tick. The inter-arrival time establishes the frequency of ticks that will see the arrival of these request groups. An average inter-arrival time of 1 means that most ticks see the arrival of requests versus larger inter-arrival times that translate into longer periods of time in which no requests arrive. Requests will stop arriving after the specified total number of requests have been generated.

All the following results are obtained from simulation runs in which the single item is unit-sized. Note that in terms of implementation each scheduling algorithm must be consulted each system tick for a decision to wait or broadcast.

4.4.2 Performance Comparisons

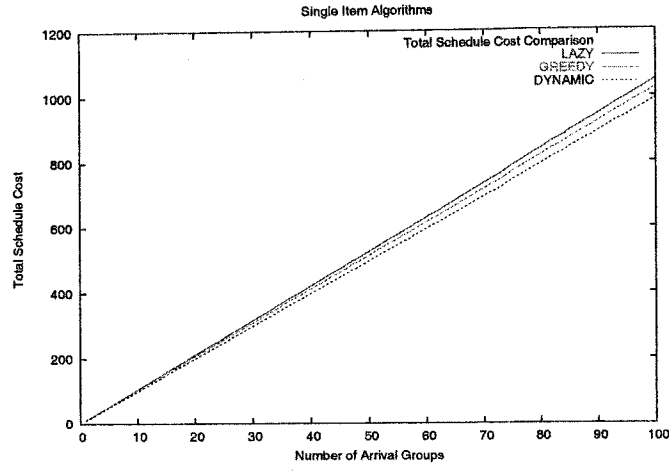
By using the optimal offline algorithm developed in Section 4.2, we are able to establish the optimal schedule for any instance. The performance of algorithms LAZY and GREEDY can then be measured by comparing the cost of schedules they produce to that of the optimal. Due to the naive decision making ability of the LAZY algorithm, we expect in general to see that it is inferior (it's schedules are more costly) to GREEDY. Likewise we expect GREEDY to be close but not equal to the optimal solution.

Figure 4.4 illustrates the performance of the three algorithms over instances generated with a $\beta = 10$, an average inter-arrival time between groups of 1, and an average group size of 5 requests. As expected, GREEDY outperforms LAZY though GREEDY is not itself optimal.

By increasing the value of β to 30 (holding all other parameters static), we can further illustrate the weakness of the LAZY algorithm. As β becomes increasingly large compared to the number of requests waiting in queue, LAZY will blindly continue to broadcast as often as possible while GREEDY compensates for the change. Compare Figure 4.5 to Figure 4.4 and note that while GREEDY remains stable in its performance relative to optimal, the degradation of LAZY's performance is significant.

While it is clear the impact of an increasing β is to increase the cost of schedules produced

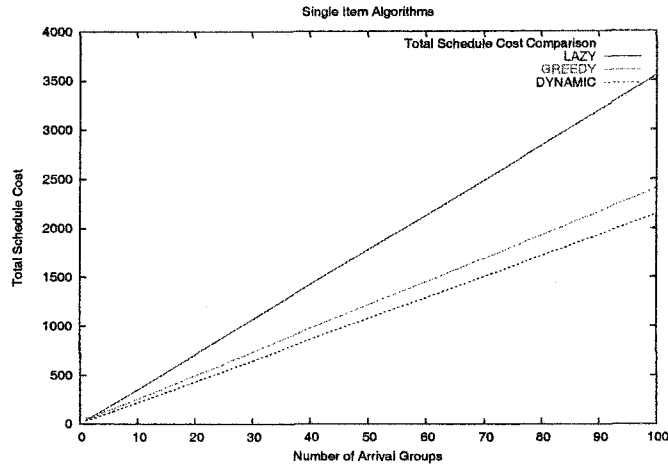
Figure 4.4: BENCHMARKING AT BETA = 10



by LAZY, note that increasing the average inter-arrival time of arriving group will mitigate this effect. Intuitively, this is due to the tendency of GREEDY to wait until a justification to broadcast is clear. A setting in which groups of requests consistently arrive in bursts is ideal for an algorithm that never hesitates to broadcast (LAZY) but troublesome for one that will wait to see if more requests will accumulate to satisfy at once. The upper graph of Figure 4.6 shows the impact of taking the parameters of Figure 4.4 and increasing the average inter-arrival time by a factor of 10. We see that the performance of GREEDY suffers while LAZY approaches optimality, a trend supported by the competitive ratio of LAZY for effectively small β values.

Finally, it can be seen that the performance of both algorithms is heavily dependant upon the size of the arriving request groups. The inherently greedy nature of both online algorithms allows rapid broadcasting when the number of requests becomes large. As mentioned in the Introduction, it is this ability of data broadcasting to scale with respect to

Figure 4.5: EFFECT OF INCREASING BETA

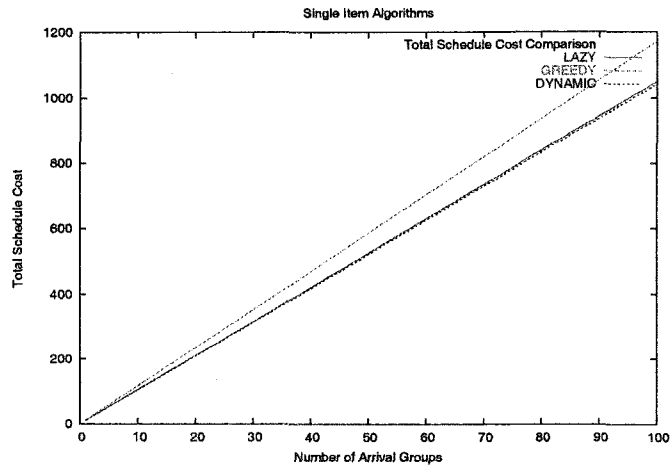


massive request loads that has made the technique popular. As the impact of a broadcast cost disappears, the single channel, single item model becomes trivially solvable online by broadcasting at every opportunity (something LAZY does at all times and GREEDY reverts to when β is negligible. The lower graph of Figure 4.6 shows the impact of taking the parameters of Figure 4.4 and increasing the average group size time by a factor of 10. The graph of all three algorithms overlap as both the LAZY and GREEDY algorithms approach optimal.

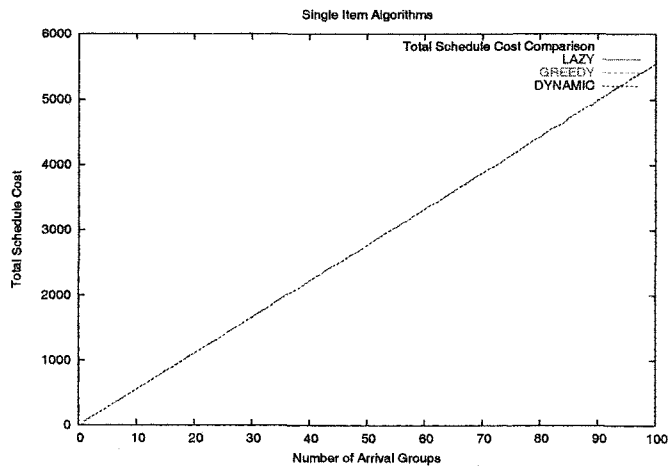
4.4.3 Simulation Conclusions

It is important to point out that most practical settings will more closely match the parameter makeup of Figure 4.4 and, as such, the GREEDY algorithm remains competitively superior to LAZY. The remaining figures in this section illustrate, however, that under extreme parameter values both algorithms may establish near-optimal performance.

Figure 4.6: EFFECTS OF MODIFYING THE INPUT SEQUENCE



INCREASING AVERAGE INTERARRIVAL TIME



INCREASING AVERAGE GROUP SIZE

Chapter 5

Minimizing Two Metrics: The Multi-Item Case

The model described in the previous chapter considered a version of the Data Broadcast Scheduling Problem in which the server attempts to minimize two metrics in the context of a single database item. We now extend that model to allow for a server with access to a database of multiple items. The resulting model described in this chapter may also be viewed as an extension of the problem described by Mao [37] which operated in a multi-item context without the second metric of broadcast cost. This chapter first formally defines the model and then proceeds to describe the analytical and simulation results of an algorithm designed for this context.

5.1 Model Description

As in the previous chapter we now consider a server attempting to maximize the quality of service as perceived by both clients and server. To this end, the server will attempt to minimize two conflicting metrics of total wait time and total broadcast cost. These metrics

are conflicting in that increasing the number of broadcasts will tend to reduce client wait time as it increases server cost and vice versa. The server has access to a database of m items and a single channel over which the items may be transmitted. From time to time requests will arrive for these items. These requests are described by a sequence of triples indicating the time at which a group of requests arrives, the number of requests in the group, and the item that will satisfy those requests. The cost of broadcasting any item in the database is β . At the beginning of each system tick, the server must decide whether or not to broadcast an item for the next tick and, if a broadcast is to be made, which item should be broadcast. That is, $(a_1, n_1, p_1), (a_2, n_2, p_2), \dots, (a_k, n_k, p_k)$ describes an input sequence where a_j is the arrival time of the j th group of requests, n_j is the number of requests in the j th group that arrived at time a_j , and p_j is the item requested by the j th group of requests. The server may refrain from broadcasting the item (referred to as “waiting”) even when requests are in queue. This is in contrast to models, such as the one described in Chapter 3, in which the server attempts only the minimization of total wait time. Unlike those models, it makes sense for a server with a broadcast cost metric to sometimes wait, even with requests in queue, when the cost of waiting is smaller than the cost of broadcasting. In order to maximize the quality of service this system provides to its clients and itself, the server will attempt to minimize the total cost of a schedule S determined by the objective function

$$OBJ(S) = \sum_{j=1}^n w_j + B \cdot \beta,$$

where the wait time of each request j , w_j , is summed over the total number of requests n and the total number of broadcasts B in schedule S is multiplied by the cost of single

broadcasting β^1 .

5.2 Algorithms

In this chapter we consider two algorithms the server may employ to minimize the above objective function, MRF and GREEDY. While GREEDY is designed for this context, MRF is not. We employ MRF here due to the lack of an optimal algorithm with which to compare the performance of GREEDY.

Most Requests First (MRF) selects for broadcast the item having the largest number of unsatisfied requests currently in queue. As an alternative description, the item selected for broadcast is the one that minimizes the incremental cost, in terms of wait time, to the schedule produced so far. Specifically, let r_i be the number of requests in the wait queue for item i , let b_t be the item selected for broadcast at time t , and let β be the uniform cost of broadcasting an item. At each tick t during which there is at least one request in the wait queue, MRF will select b_t such that $b_t \in 1 \dots m$, $r_{b_t} > 0$, and b_t minimizes the incremental cost $\sum_{i \in 1 \dots m, i \neq b_t} r_i + \beta = \sum_{i \in 1 \dots m} r_i - r_{b_t} + \beta$. Because the broadcast cost is the same for all items, the item selected for broadcast will be the item with the largest number of unsatisfied requests.

As an example, consider a $\beta = 20$ and an input sequence $(0, 5, 1)$, $(0, 5, 2)$, $(0, 1, 3)$, $(1, 1, 1)$, $(1, 8, 2)$, $(1, 5, 3)$, $(2, 2, 1)$, $(2, 10, 2)$, $(2, 8, 3)$. MRF constructs the schedule in Figure 5.1 with a total wait time of 74, a total broadcast cost of 100, and a total schedule cost of 174.

¹Because broadcast cost and wait time are measured in different units, some weighted measure between the two is desired. Thus the objective function might be written as $\alpha(\sum_{j=1}^n w_j) + B \cdot \beta'$. However, for simplicity β' can be normalized to β , the relative cost of broadcasting relative to a unit of wait time.

Figure 5.1: EXAMPLE SCHEDULING DIAGRAM OF MOST REQUESTS FIRST (MRF)

		Time				
		0	1	2	3	4
Items	1	5	1	3	3	3
	2	5	13	10	10	
	3	1	6	14		

c = 2 using MRF

Given that a broadcast should be made, we expect MRF to make a reasonable decision as to which item should be selected, however we note that MRF will always broadcast if there are any requests unsatisfied. That is, MRF naively does not consider the impact of broadcast cost. A smarter online approach than that above is to determine if the wait time saved by a broadcast is worth the cost of making it. The GREEDY algorithm attempts to make such a good broadcasting decision by considering both the broadcast cost β and the amount of accumulated wait time of all unsatisfied requests. It compares the amount of wait time unsatisfied requests have accumulated while in queue to the cost of satisfying them. If β is large in comparison to this total accumulated wait time, the algorithm may choose to wait until such a time as the cost of broadcasting is more warranted.

More specifically, let t be the time at which GREEDY must make a decision to wait or broadcast at the next time $t + 1$. Let P_i be the number of pending (unsatisfied) requests for item i that have arrived up to time t , inclusively, since the last broadcast was made. Let Q_i be the accumulated wait time of all P_i pending requests at time $t + 1$ by all P_i pending requests for item i . At each decision point, if the wait time accumulated by the requests waiting for any item is no greater than the broadcast cost, then the server will choose not

to broadcast at the next opportunity, otherwise it will broadcast that item requested by the group of requests whose accumulated wait time has exceeded the broadcast cost. The pseudocode for GREEDY is given below.

```

GREEDY Online Algorithm
Single broadcast cost: beta
Input request sequence: (a[1],n[1][1...m],...,a[k],n[k][1...m])
i.e. n[1][j] is the number of requests arriving at a[i] for item j

do
t = a[1]
for i <- 1 to m
    P[i] <- n[1][i]      // # requests for item i at a[1]
    Q[i] <- n[1][i]      // each request accumulates 1 unit of wait time at a[i]+1
repeat
maxQ <- max{Q[i]}
maxI <- index of maxQ
if beta <= maxQ
then broadcast item maxI at t+1
    Q[maxI] <- 0
    P[maxI] <- 0
    for i <- 1 to m
        if i <> maxI
            Q[i] <- Q[i] + P[i]
else wait at t+1
    for i <- 1 to m
        Q[i] <- Q[i] + P[i]
t <- t+1
if t = a[j] for some j
then for i <- 1 to m
    Q[i] <- Q[i] + n[j][i]
    P[i] <- P[i] + n[j][i]
until all requests have arrived and have been satisfied

```

As an example, consider a $\beta = 20$ and an input sequence $(0, 5, 1)$, $(0, 5, 2)$, $(0, 1, 3)$, $(1, 1, 1)$, $(1, 8, 2)$, $(1, 5, 3)$, $(2, 2, 1)$, $(2, 10, 2)$, $(2, 8, 3)$. GREEDY constructs the schedule in Figure 5.2 with a total wait time of 111, a total broadcast cost of 60, and a total schedule

Figure 5.2: EXAMPLE SCHEDULING DIAGRAM OF GREEDY

		Time				
		0	1	2	3	4
Items	1	5	6	8	8	8
	2	5	13	23		
	3	1	6	14	14	

c = 2 using GREEDY

cost of 171.

5.3 Analytical Results

5.3.1 A Lower Bound for GREEDY

Theorem 5.1 *For any instance let C be the total cost of the schedule produced by MRF and C^* be the total cost of the optimal schedule. Then, $C \leq 2C^*$.*

Proof: We prove the lower bound by establishing that there is at least one instance for which the total wait time of the schedule produced by GREEDY is equal to 2 times that of the OPT schedule.

We construct this instance as follows. At time $t = 0$ all n requests will arrive, divided evenly among the m items. That is, let $n_1 = \frac{n}{m}$ resulting in the arrival sequence $(0, n_1, 1), \dots, (0, n_1, m)$. This instance causes GREEDY to wait needlessly where a more optimal solution would begin broadcasting items immediately.

Greedy will choose not to broadcast any item until the accumulated wait time of all requests for some item has exceeded β . It is clear, then, that GREEDY will desire to

broadcast all items at the same point in time, say $t = l$. Due to the limitations of the single channel, however, GREEDY will not be able to satisfy all requests until m ticks later. More formally, let l be the tick at which the first broadcast is made. This means that m sets of n_1 requests have waited for the previous $l - 1$ ticks. Thus, a total wait time of $m(l - 1)n_1$ has accrued before a broadcast is made. The group of requests satisfied by the first broadcast will incur one additional tick of waiting, the group satisfied by the second broadcast will incur two additional ticks of waiting and so on. As such, $1n_1 + 2n_1 + \dots + mn_1 = \frac{1}{2}m(m+1)n_1$ total wait time will accrue after the broadcasting of items has begun. Clearly m broadcasts will be needed resulting in a total broadcast cost of $m\beta$. The total cost of the schedule produced by GREEDY in this instance is then

$$C = \frac{1}{2}m(m+1)n_1 + m(l-1)n_1 + m\beta.$$

In contrast to the behavior of GREEDY above, the OPT schedule begins broadcasting immediately, satisfying a group of requests each tick. That is, the schedule produced by OPT eliminates the l ticks GREEDY waiting before its first broadcast resulting in a schedule that satisfies all requests within m ticks for a cost of

$$C^* = \frac{1}{2}m(m+1)n_1 + m\beta.$$

Without loss of generality, assume $n_1 = \frac{\beta}{l}$. From the costs of each schedule described above the ratio of GREEDY over OPT is then

$$\begin{aligned}
\frac{C}{C^*} &= \frac{\frac{1}{2}m(m+1) + m(l-1)n_1 + m\beta}{\frac{1}{2}m(m+1) + m\beta} \\
&= \frac{\frac{1}{2}m(m+1)\frac{\beta}{l} + m(l-1)\frac{\beta}{l} + m\beta}{\frac{1}{2}m(m+1)\frac{\beta}{l} + m\beta} \\
&= \frac{m(m+1)\beta + 2m(l-1)\beta + 2m\beta l}{m(m+1)\beta + 2m\beta l} \\
&= 1 + \frac{2m(l-1)\beta}{m(m+1)\beta + 2m\beta l} \\
&= 1 + \frac{2(l-1)}{(m+1) + 2l} \\
&= 1 + \frac{2l-2}{2l+m+1}
\end{aligned}$$

As $l \rightarrow \infty$, meaning that β has become arbitrarily large, the above ratio approaches 2. ■

5.4 Simulation Results

In order to develop a feel for the empirical behavior of each algorithm, we utilize a discrete-event simulation to model the effects of various input sequence properties on the performance of MRF and GREEDY. Unlike the model of the previous chapter in which finding an optimal solution was polynomial, the model in this chapter is computationally more strenuous. For arbitrarily large instances of the problem it is time-consuming to determine the extent to which a schedule produced by MRF or GREEDY deviates from the optimal. As such, we compare the performance of each algorithm strictly to the other.

For the results to follow, each replication (simulated instance) consists of stochastically generating an arrival sequence, simulating the server's scheduling decisions according to

the algorithm under study, and collecting data to compute the sum of wait times, sum of broadcast cost, and total schedule cost statistics. Because the terminal condition is specified by the number of processed requests, the terminal time (makespan) will vary from instance to instance. Large numbers of replications (1000 replications for results displayed in figures) are used for each parameter change to ensure that the behavior observed is not dependent on any one run. Thus, replications are generated such that statistics are reported at 95% confidence.

5.4.1 Simulation Parameters

Several parameters to the problem will affect the characteristics of the input sequence that is simulated on each algorithm for each replication: the arrival rate, the inter-arrival time, and the total number of requests set to arrive. In the absence of motivation for any other distribution, the arrival rate of requests is generated through use of a discrete version of the Exponential distribution. This rate measures the number of requests, or the size of the group of requests, arriving per tick. The inter-arrival time establishes the frequency of ticks that will see the arrival of these request groups. An average inter-arrival time of 1 means that most ticks see the arrival of requests versus larger inter-arrival times that translate into longer periods of time in which no requests arrive. Requests will stop arriving after the specified total number of requests have been generated.

All the following results are obtained from simulation runs in which the three items in the database are unit-sized. Note that in terms of implementation each scheduling algorithm must be consulted each system tick for a decision to wait or broadcast.

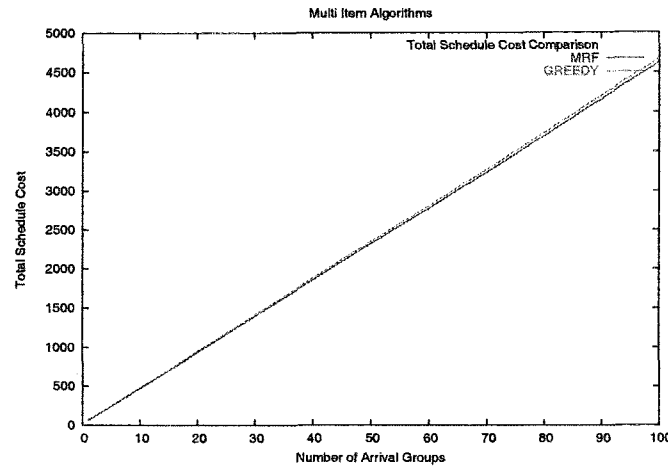
5.4.2 Performance Comparisons

Due to the fact that MRF does not consider broadcast cost in its decision making, we expect in general to see that it is inferior (it's schedules are more costly) to GREEDY within contexts where the broadcast cost plays a large role. MRF does aggressively minimize wait time, however, whereas GREEDY is more careful to avoid unnecessary broadcast cost. When the value of β is relatively small and there are many requests in queue, it can be seen that MRF performs quite well, since any unnecessary broadcasts are not remarkably harmful to the overall schedule cost. When β is small, GREEDY may also become too reactive - accumulating wait time in an attempt to avoid the minimal penalty of broadcasting too readily.

Figure 5.3 illustrates the performance of the two algorithms over instances generated with a $\beta = 10$, an average inter-arrival time between groups of 1, and an average group size of 5 requests. While the algorithms do show comparable performance, it is clear that MRF outperforms GREEDY under this set of parameters.

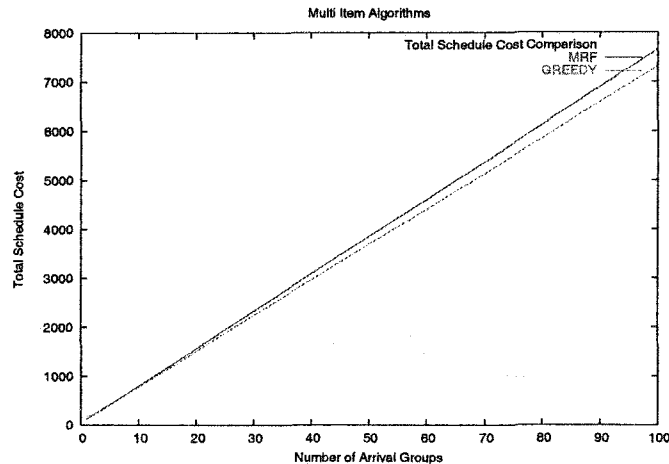
As stated earlier, however, the strength of GREEDY (and the weakness of MRF) lies in its ability to compensate for broadcast costs that are large enough to warrant waiting for a more justifiable time to broadcast an item. By increasing the value of β to 30 (holding all other parameters static), we can illustrate the weakness of MRF in adapting to large broadcast costs. As β becomes increasingly large compared to the number of requests waiting in queue, MRF will blindly continue to broadcast as often as possible while GREEDY compensates for the change. Compare Figure 5.4 to Figure 5.3 and note that GREEDY now outperforms MRF.

Figure 5.3: BENCHMARKING AT BETA = 10



While it is clear the impact of an increasing β is to diminish the effectiveness of MRF, note that increasing the average inter-arrival time of arriving groups will likewise mitigate the effectiveness of GREEDY. Intuitively, this is due to the tendency of GREEDY to wait until a justification to broadcast is clear. A setting in which groups of requests consistently arrive in bursts is ideal for an algorithm that never hesitates to broadcast (MRF) but troublesome for one that will wait to see if more requests will accumulate to satisfy at once. The top graph of Figure 5.5 shows the impact of taking the parameters of Figure 5.3 and increasing the average inter-arrival time by a factor of 10. We see that the performance of GREEDY suffers while MRF continues to perform well, a trend supported by the competitive ratio of MRF for effectively small β values.

Finally, it can be seen that the performance of both algorithms relative to each other is largely independent of the size of the arriving request groups when beta is relatively small. For MRF, the sheer size of each group is not as important as the size of each group with

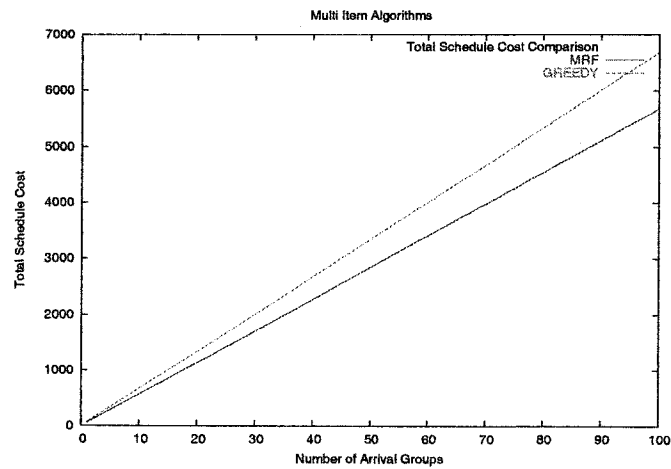
Figure 5.4: EFFECT OF INCREASING BETA to 30

respect to the others and for GREEDY large group sizes are relevant only in comparison to β . As mentioned in Chapter 1, it is this ability of data broadcasting to scale with respect to massive request loads that has made the technique popular. The bottom graph of Figure 5.5 was created using the same parameters as those of Figure 5.3. Note that while the scale of total schedule cost is much larger in the bottom graph of Figure 5.5, the performance of both algorithms relative to each other is steadily analogous to that 5.3.

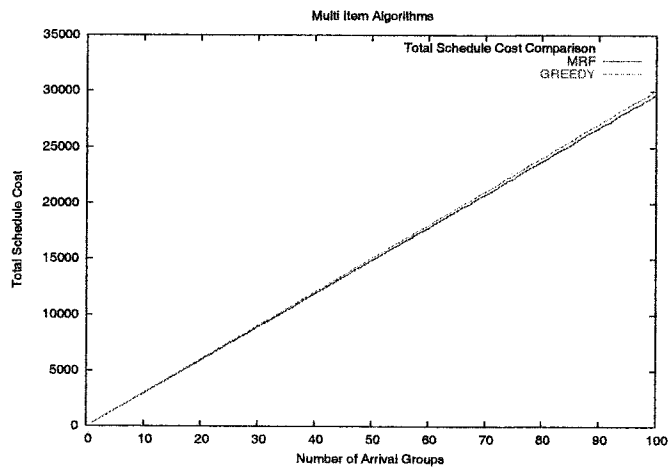
5.4.3 Simulation Conclusions

While a quick glance at the figures above would seem to show more instances in which MRF outperforms GREEDY, it is important to note that the settings in which MRF did excel were those minimizing the impact of the broadcast cost. In most real settings, the use of a broadcast cost will naturally imply that the cost is non-trivial. As such, the superior performance of GREEDY over MRF when broadcast costs are large should be emphasized.

Figure 5.5: EFFECTS OF MODELING THE INPUT SEQUENCE



INCREASING AVERAGE INTERARRIVAL TIME



INCREASING AVERAGE GROUP SIZE

Chapter 6

Conclusions and Future Work

In this thesis we have contributed results to three different models of the data broadcast scheduling problem. In data broadcasting, a server with access to a set of items attempts to satisfy data requests from its clients by efficiently broadcasting those items. The question of what item to send at each opportunity is addressed by a scheduling algorithm whose performance is of critical concern. In order to evaluate the performance of an algorithm in a given instance, the schedule constructed is measured by an objective function representing the quality of service provided by the system to its clients and, at times, to itself. While most of the work done to date relies on stochastic assumptions to analyze these scheduling algorithms, we employ a worst-case technique known as competitive analysis. The significance of our approach is that it provides solid measures of performance over all instances in contrast to those based on historical distributions that may not be accurate or available. As broadcasting systems may attempt to optimize various aspects of performance, a specific broadcasting model will be distinguished by differing objective functions, and thus by differing characteristics of metric optimization, number of database items, and number of broadcast channels. In this thesis we have addressed three such models :

1. In Chapter 3 we have described the data broadcast scheduling problem in which

a server attempts to minimize the total wait time of all requests in the context of a single channel and multiple database items. This model is of interest because it is well-known in the field and represents the most commonly used metric in broadcasting. In this context, we have established competitive ratios for the on-line algorithms First Come First Served (FCFS) and Most Requests First (MRF). Like the model in which they are found, these two algorithms are well-utilized benchmarks. We have also provided a general lower bound for all algorithms in this context. These contributions establish that both FCFS and MRF have a performance in the worst-case that is as good as any algorithm can hope to achieve. At the same time we have described simulation experiments that compare the performance of MRF to FCFS using assumptions of exponential arrival rate and equally likely item distributions. These experiments indicate that MRF is superior to FCFS on average, despite their equal competitive ratios in the worst-case.

2. In Chapter 4 we have introduced a model that, to our knowledge, has never appeared in the pull-based¹ version of data broadcasting. We consider a server with the goal of minimizing both the total wait time and total broadcast cost in the context of a single channel and single database item. Broadcast cost is a useful metric that measures the “drain” of broadcasting on the system itself. This metric is desirable because it is highly versatile and can be used to represent many features of relatively new systems (such as the battery drain in mobile devices). It’s inclusion in this model is also of interest because the objective function is then one composed of two conflicting met-

¹As discussed in Chapter 2, pull-based broadcasting, in which the server is explicitly aware of client requests, contrasts with push-based broadcasting, in which the server assumes the requests follow some distribution.

rics. We have described two algorithms (LAZY and GREEDY) specifically developed for this model and established their competitive ratios. In addition, we have developed an optimal off-line algorithm for this context which has been used to simulate the performance of the two on-line algorithms using assumptions of exponential arrival rate and equally likely item distributions. These simulations indicate that while both algorithms may achieve near optimal performance under certain circumstances, GREEDY is competitive superior to LAZY under more realistic settings.

3. In Chapter 5 we have extended the model from Chapter 4 to minimize both the total wait time and total broadcast cost in the context of a single channel and multiple database items. This extension pushes the model toward greater applicability and complexity. We have established a lower bound to the GREEDY algorithm. Again, the use of broadcast cost in this pull-based context is new to the field. For lack of an optimal algorithm in this context, we have compared the performance of the GREEDY algorithm to that of MRF. We have described simulation experiments using assumptions of exponential arrival rate and equally likely item distributions that compare the performance of MRF to GREEDY. While MRF does outperform GREEDY in many settings, those settings minimize the impact of the broadcast cost. In instances where broadcast cost is significant we have shown the GREEDY algorithm is superior on average.

In addition to the analytical results above we have accompanied the analysis in each chapter by scheduling diagrams and simulation experiments that provide a backdrop for the competitive ratios supplied. It should be noted that the graphical depiction of the pull-based

model we have engendered is an original tool with the potential to greatly enhance pull-based model discussions and there are several general reasons we have incorporated simulation results other than those mentioned specifically above. First, while we emphasize worst-case performance it is important to note that this measure is not meant to supersede average case analysis but to complement it (or provide some measure when no average case assumptions are reliable). In terms of algorithm design, our contributions have established how well these above algorithms perform in worst-case scenarios, and it is useful to see examples of how that performance compares to what a server might see on average. Secondly, the models incorporating broadcast cost have not been studied in the pull-based version of the problem and thus have no previous simulations with which to compare their performance. Thirdly, other than the model of Chapter 4 that we have shown to be solvable in polynomial time, the other models deal with scheduling problems that are NP-Hard (See Chapter 2 for discussion). As such, it is computationally infeasible to provide direct comparisons of an algorithm to the optimal and we employ simulation to provide some relative comparison of the algorithms in question to each other. This computational intractability has implications for how data broadcasting models of additional complexity will be pursued in the future.

On that note we can envision several directions worthy of additional pursuit. These areas of future work fall broadly under the two categories model extensions and empirical research.

While the push-based community has studied an array of metrics and system characteristics, many of these model extensions have not yet been exhausted analytically within the pull-based system environment. From the perspective of the system environment, the use of multi-sized database items [42] [15] introduces interesting questions of allowing the server to

preempt the transmission of an item before its completion as well provides the challenge of creating algorithms capable of dealing with this additional option. The concept of request satisfaction would need to be redefined as the start of a broadcast would no longer be a sufficient condition of satisfaction. With respect to request characteristics the application of non-infinite deadlines with penalties would add an interesting model variation. A new definition of system termination would need to be established to provide for requests that remain forever unsatisfied.

The increasing complexity of these models will test the limits of the worst-case analysis techniques we employ and simple simulation analysis may not be able to construct optimal schedules for empirical studies of real-world size. As discussed in Chapter 2, the NP-hardness of several models has already been established and there is reason to believe more complicated model extensions will also fall into the category of NP-hard. Thus, as seen in Chapter 3, it becomes computationally infeasible to compare the performance of heuristics to optimal solutions when instance sizes are large and the optimal solutions are exponential calculations. For settings in which optimal solutions are required for performance measures, it is likely that techniques enabling more efficient searches in combinatorial spaces (such as constraint processing discussed at length by Dechter [18]) will need to be applied to the data broadcast scheduling problem.

Bibliography

- [1] S. ACHARYA. Balancing push and pull for data broadcast. In *Proceedings ACM SIGMOD*, Tuscon, Arizona, 1997.
- [2] S. ACHARYA, R. ALONSO, M. FRANKLIN, AND S. ZDONIK. Broadcast disks: Data management for asymmetric communication environments. Technical Report CS-94-43, Brown University, Providence RI, 1994.
- [3] S. ACHARYA, M. FRANKLIN, AND S. ZDONIK. Dissemination-based data delivery using broadcast disks. In *IEEE Personal Communications*, pages 2(6):50–60, Roma, Italy, 1995.
- [4] S. ACHARYA, M. FRANKLIN, AND S. ZDONIK. Dissemination-based data delivery using broadcast disksprefetching from a broadcast disk. In *Proceedings of the 12th International Conference on Data Engineering*, 1996.
- [5] S. ACHARYA AND S. MUTHUKRISHNAN. Scheduling on-demand broadcasts: new metrics and algorithms. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile COmputing and Networking (Mobicom)*, pages 43–54, 1999.
- [6] D. AKSOY, M. ALTINEL, R. BOSE, U. CETINTEMEL, M. FRANKLIN, J. WANG, AND S. ZDONIK. Research in data broadcast and dissemination. In *Proceedings of the 1st International Conference on Advanced Multimedia Content Processing*, 1998.
- [7] D. AKSOY AND M. FRANKLIN. On-demand broadcast scheduling. Technical Report CS-TR-3859, University of Maryland, College Park MD, 1998.
- [8] D. AKSOY AND M. FRANKLIN. Scheduling for large-scale on-demand data broadcasting. In *Proceedings of the IEEE INFOCOM Conference*, pages 651–659, 1998.
- [9] D. AKSOY, M. FRANKLIN, AND S. ZDONIK. Data staging for on-demand broadcast. In *Proceedings of the 27th VLDB Conference*, pages 651–659, Roma, Italy, 2001.
- [10] M. H. AMMAR AND J. W. WONG. Analysis of broadcast delivery in a videotext system. In *IEEE Transactions on Computers*, pages 34(9):863–966, 1985.
- [11] M. H. AMMAR AND J. W. WONG. The design of teletext broadcast cycles. In *Performance Evaluation*, pages 5(4):235–242, 1985.
- [12] M. H. AMMAR AND J. W. WONG. On the optimality of cyclic transmission in teletext systems. In *IEEE Transactions on Communications*, pages 35:68–73, 1987.

- [13] A. BAR-NOY, R. BHATIA, J. NAOR, AND B. SCHIEBER. Minimizing service and operation costs of periodic scheduling (extended abstract). In *Symposium on Discrete Algorithms*, pages 11–20, 1998.
- [14] A. BAR-NOY, B. PATT-SHAMIR, AND I. ZIPER. Broadcast disks with polynomial cost functions. In *INFOCOM (2)*, pages 575–584, 2000.
- [15] Y. BARTAL AND S. MUTHUKRISHNAM. Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 558–559, 2000.
- [16] J. L. BENTLEY AND C. MCGEOCH. Amortized analysis of self-organizing sequential search heuristics. *Communication of the SCM*, 28(4):404–411, 1985.
- [17] A. BORODIN AND R. EL-YANIV. *Online computation and competitive analysis*. Cambridge University Press, first printing edition, 1998.
- [18] R. DECHTER. *Constraint processing*. Morgan Kaufmann, first printing edition, 2003.
- [19] H. D. DYKEMAN, M. H. AMMAR, AND J. W. WONG. Scheduling algorithms for videotex system under broadcast delivery. In *Proceedings of the International Conference on Communications*, pages 1847–1851, 1986.
- [20] T. ERLEBACH AND A. HALL. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 194–202, San Francisco, California, 2002.
- [21] M. R. GAREY AND D. S. JOHNSON. *Computers and intractability*. New York, W. H. Freeman, twenty-first printing edition, 1979.
- [22] A. V. GOLDBERG, H. KAPLAN, AND R. E. TARJAN. Heaps with time, applied to broadcast scheduling, manuscript, 1999.
- [23] R. L. GRAHAM. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
- [24] Y. GUO, S. K. DAS, AND C. M. PINOTTI. A new hybrid broadcast scheduling algorithm for asymmetric communication systems: Push and pull data based on optimal cut-off point. In *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 123–130. ACM Press, 2001.
- [25] S. HAMEED AND N. H. VAIDYA. Log-time algorithms for scheduling single and multiple channel data broadcast. In *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 90–99, 1997.
- [26] S. HAMEED AND N. H. VAIDYA. Efficient algorithms for scheduling data broadcast. *Wireless Networks*, 5(3):183–93, 1999.

- [27] A. T. HAWKINS AND W. MAO. On multi-channel data broadcast scheduling. In *Proceedings of the 6th Joint Conference on Information Sciences (JCIS)*, pages 915–918, Research Triangle Park, North Carolina, March 2002.
- [28] S. JIANG AND N. H. VAIDYA. Scheduling algorithms for a data broadcast system: Minimizing variance of the response time. Technical Report TR 98-005, Texas A&M University, College Station, TX, 1998.
- [29] B. KALYANASUNDARAM AND K. PRUHS. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.
- [30] B. KALYANASUNDARAM, K. PRUHS, AND M. VELAUTHAPILLAI. Scheduling broadcasts in wireless networks. In *European Symposium on Algorithms*, 2000.
- [31] A. R. KARLIN, M. S. MANASSE, L. RUDOLPH, AND D. D. SLEATOR. Competitive snoopy aaching. *Algorithmica*, 3(1):79–119, 1988.
- [32] R. M. KARP. On-line algorithms versus off-line algorithms: How much is it worth to know the future? Technical Report TR-92-044, International Computer Science Institution, 1992.
- [33] C. KENYON AND N. SCHABANEL. The data broadcast problem with non-uniform transmission times. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999.
- [34] C. KENYON, N. SCHABANEL, AND N. YOUNG. Polynomial-time approximation scheme for data broadcast. In *ACM Symposium on Theory of Computing*, pages 59–66, 2000.
- [35] E. KOUTSOUPIAS AND C. H. PAPADIMITRIOU. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science*, pages 394–400, Santa Fe, New Mexico, 20–22 November 1994.
- [36] M. S. MANASSE, L. A. MCGEOCH, AND D. D. SLEATOR. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 322–333, Chicago, Illinois, May 1988.
- [37] W. MAO. Competitive analysis of on-line algorithms for on-demand data broadcast scheduling. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 292–296, 2000.
- [38] S. HAMEED N. H. VAIDYA. Scheduling data broadcast in asymmetric communication. Technical Report TR-96-022, Texas A&M University, College Station, TX, 1996.
- [39] J. OH, K. HUA, AND K. PRABHAKARA. A new broadcasting technique for an adaptive hybrid data delivery in wireless mobile network environment.
- [40] S. PHILLIPS AND J. WESTBROOK. On-line algorithms: Competitive analysis and beyond. In *Algorithms and Theory of Computation Handbook*, M. J. Atallah, editor, chapter 10. CRC Press, Boca Raton, 1999.

- [41] M. PINEDO. *Scheduling theory, algorithms and systems*. Prentice Hall, second edition edition, 2001.
- [42] J. EDMONDS K. PRUHS. Broadcast scheduling: When fairness is fine. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 421–430, 2002.
- [43] N. SCHABANEL. The data broadcast problem with preemption. In *Symposium on Theoretical Aspects of Computer Science*, pages 181–192, 2000.
- [44] D. D. SLEATOR AND R. E. TARJAN. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
- [45] O. ULUSOY AND M. KARAKAYA. Evaluation of a broadcast scheduling algorithm. In *Advances in Databases and Information Systems, 5th East European Conference*, pages 182–195, 2001.
- [46] N. H. VAIDYA AND S. HAMEED. Data broadcast scheduling: on-line and off-line algorithms. Technical Report TR96-017, Texas A&M University, College Station, TX, 29 1996.
- [47] R. GANDHI S. KHULLER Y. KIM Y. WAN. Algorithms for minimizing response time in broadcast scheduling. In *Proceedings of the Ninth Conference on Integer Programming and Combinatorial Optimization*, 2002.
- [48] J. W. WONG. Broadcast delivery. In *Proceedings of the IEEE*, pages 1566–1577, 1988.
- [49] N. YOUNG. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 241–250, San Francisco, California, 28–30 January 1991.
- [50] N. E. YOUNG. Competitive paging and dual-guided on-line weighted caching and matching algorithms. Technical Report 348–91, Ph. D. Thesis, Department of Computer Science, Princeton University, 1991.
- [51] N. E. YOUNG. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.
- [52] N. E. YOUNG. On-line file caching. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, San Francisco, California, 25–27 January 1998.

VITA

Aaron Thomas Hawkins

Aaron Thomas Hawkins was born in Henderson, North Carolina on November 4th, 1976 and graduated from Mountain Heritage High School in Burnsville, North Carolina in 1999. He earned a Bachelor of Science degree in both Mathematics and Computer Science from Mars Hill College in May 1999. He received a Master of Science degree in Computer Science from the College of William and Mary in May, 2001, and will receive a Doctor of Philosophy degree in Computer Science in May 2005. The author works for the Intelligent Systems and Planning division of Rockwell Scientific in the Research Triangle, North Carolina.