Who, When, Where: Timeslot Assignment to Mobile Clients

Fangfei Chen*, Matthew P. Johnson†, Yosef Alayev†, Amotz Bar-Noy† and Thomas F. La Porta*

*Department of Computer Science and Engineering, The Pennsylvania State University
†Department of Computer Science, The City University of New York Graduate Center

Abstract—We consider variations of a problem in which data must be delivered to mobile clients en-route, as they travel toward their destinations. The data can only be delivered to the mobile clients as they pass within range of wireless base stations. Example scenarios include the delivery of building maps to firefighters responding to multiple alarms. We cast this scenario as a parallel-machine scheduling problem with the little-studied property that jobs may have different release times and deadlines when assigned to different machines. We present new algorithms and also adapt existing algorithms, for both online and offline settings. We evaluate these algorithms on a variety of problem instance types, using both synthetic and real-world data, including several geographical scenarios, and show that our algorithms produce schedules achieving near-optimal throughput.

I. INTRODUCTION

Consider a scenario in which mobile clients are traveling along routes within a geographic region, towards destinations at which they have a mission to complete. Upon reaching their destinations, they will require instructions or resources (data items) in order to complete their missions. Since it may be difficult for various reasons to ensure that each client is given its data item at its time of departure, data items are delivered to the mobile clients en-route as they pass within range of wireless base stations (BSs). The BSs have high-speed communication links to the wired network and are dispersed throughout the region (see Figure 1).

Since the wireless link has limited range, a client must obtain its data item from one of the BSs lying sufficiently near to its route. Assuming sufficient bandwidth in the wired portion of the network, the bottleneck of the system lies in the limited bandwidth between BSs and mobile clients. The time to transfer the data depends on the data item size and the channel’s transmission rate. Because clients are moving, there will be a limited time window (possibly empty) in which they are able to receive data from each particular BS. In order for a given client to succeed, therefore, its data item must be scheduled for download from some BS, during the feasible time window in which the client is in its range. If a given client approaches only a few BSs or if many clients approach the same BSs, it may be difficult to satisfy all clients. Our goal is to maximize the (possibly weighted) number of successful clients.

This problem can be recast as a parallel-machine scheduling problem. In this interpretation, the wireless links of BSs play the role of machines (with multiple, co-located machines corresponding to the channels of a BS), and the requested data items are the jobs. The scheduling problem with machine-dependent time windows is little studied, yet a growing number of applications requiring that a service be provided to mobile devices. One example is a traveling cell phone user requesting to download a multimedia file.

This problem lies within a large family of scheduling problems in which $n$ jobs (each with weight $w_j$ and processing time $p_j$, and release time $r_j$ and deadline $d_j$) are to be assigned to $m$ parallel machines [1]. A valid assignment of job $j$ to machine $k$ is to dedicate $k$ (exclusively to $j$) over some interval $[s, s+p_j)$, where $[s, s+p_j) \subseteq [r_j, d_j)$. The goal is throughput maximization, i.e., maximizing the weight of assigned jobs. In some existing settings, job sizes and weights are machine-dependent, i.e., their values depend on the machine to which the job is assigned. A crucial aspect of our problem, however, is that jobs’ release times and deadlines are machine-dependent as well, due to the clients’ mobility. This generalization has received little attention in the past but is an essential aspect of our motivating applications.

Although the problem without machine-dependent time windows is already NP-hard [1], it is well studied and has some known approximation algorithms. We will cover these algorithms in the Related Work section. In this paper, we adapt existing algorithms for related scheduling problems and also propose new algorithms, for multiple machine-dependent times settings including offline and online, which we then evaluate with synthetic data sets as well as real-world data sets obtained from UMass [2].

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

The rest of this paper is organized as follows. Section II presents related work and Section III introduces two system architectures. Section IV formally defines the problem and presents the algorithms we adopt and propose, which are then evaluated in Section V. Section VI explores the case in which data has machine-dependent weight. Finally, Section VII concludes the paper.

II. RELATED WORK

The problem we study concerns mobile data access [3]. In a mobile computing environment, users move about, carrying portable computing devices with wireless communication capability. As they travel, they require interaction with a common service provider, an access point or base station (BS), in order to access information or receive service. Because the users are moving and using wireless connections, this paradigm differs from traditional information access paradigms in that for us, connections are unstable and resources (bandwidth and channels) are limited and unshareable. The challenge is how to allocate this resource to maximize the total profit.

A Content Distribution Network [4] consists of a number of distributed servers whose job is to reduce traffic from data origin servers by delivering content to nearby users. A BS in our problem plays a similar role, viz., delivering information to passing clients. In contrast, however, data items for us are jobs to be scheduled, whereas in a CDN data items are cached in nodes indefinitely.

Our mobile dissemination setting contrasts with the classical data dissemination approach of Directed Diffusion [5], in which requested data is delivered by a multi-hop wireless connection. In our setting, we assume fast access of the base stations to the data items and focus exclusively on the last hop, i.e., from chosen base station to client. Moreover, latency does not come into our problem: a data item either gets scheduled in a feasible window before the client reaches its destination or it does not.

There is a very large literature on algorithms for scheduling jobs on parallel machines. See [1] for an introduction. Here we refer to some of the most relevant existing work. Lee et al. [6] is the primary antecedent to our work. Their problem, the unrelated machines scheduling problem (USP), minimizes the total weighted flow time, subject to time-window job availability and machine downtime constraints. In USP, job sizes, as well as release times and deadlines, are machine-dependent. Deciding whether it is possible to schedule all jobs is strongly NP-complete [7], even for the case of a single machine and even if only two integer values exist for release times and deadlines [7]. Algorithms are given [6], however, for constrained settings, as well as a zero-one integer programming (IP) formulation, which is solved with branch-and-bound techniques. Lee et al. [6] claim to be the first to study algorithmically scheduling problems with machine-dependent release times and deadlines. We are aware of very little additional work done in the interim. One recent paper on parallel-machine scheduling [8] considers release times and deadline times to be both job- and machine-dependent. The authors give heuristics only, based on a constraint programming/tabu search hybrid.

Some of the algorithms we implement are drawn from work on scheduling on identical or unrelated machines. Bar-Noy et al. [9] studies several such settings, obtaining algorithms with the following guarantees: a 2-approximation for unrelated machines and a \( \frac{1}{(1+1/m)^{m-1}} \) (which approaches \( e/(e-1) \) as the number of machines \( m \to \infty \)) for identical machines. The algorithm in both cases is called \( m \)-Greedy (in our notation), which applies an order-by-end-time greedy algorithm machine-by-machine. This algorithm can also be applied to the machine-dependent times setting.

A stack-based Two Phase algorithm with somewhat better performance is given by Berman and DasGupta [10]. Similar algorithms are given by Bar-Noy et al. [9]. These problems reduce job scheduling problems, with sizes, release times and deadlines, to the interval scheduling problem, by replacing a job with all possible (discrete) intervals in which the job can be scheduled. Such intervals are then referred to as job instances.

In online settings, jobs arrive over time and no information about a job is given before its release time. Koren et al. [11] give an algorithm allowing preemption (a job may be interrupted and resumed later) which achieves the optimal competitive ratio \( (1 + \sqrt{\kappa})^{-2} \), where \( \kappa \) is the ratio between the highest weight density and the lowest weight density (a job’s weight density is its weight divided by processing time). If preemption is forbidden, then no constant competitive ratio is possible, even if jobs have fixed start and end times [12]. For the setting of equal length jobs, however, Ding et al. [13] give an \( e/(e-1) \approx 1.582 \) competitive ratio algorithm.

III. SYSTEM ARCHITECTURE

In this section we describe our environment and discuss two general system architectures on which we base our solutions: centralized and distributed.

In the system we have base stations deployed within a geographical region and mobile clients with information needs that are traveling towards a mission site. We call the time period during which a client can talk to the BS the contact time window. A client must retrieve its information from any one of the BS within this time window. The window is decided by the speed vector, the route and the relative location of the BS and the client. Thus, each BS/client pair may have a unique contact time window. For example, in Figure 2, the client enters communication range of the BS at time \( t_1 \) and leaves at \( t_2 \), so the contact time window duration is \( t_2 - t_1 \).

Each data item (or job) has a size, and each BS has a transmission rate. The time required to complete the transmission of a data item depends on these two values. Again, this time
varies by BS/client pair, and may even change over time. We call this duration the processing time.

The system has slotted time, so the task of the system is to decide how we allocate these timeslots to different clients. There are three types of problem settings we consider:

1) Offline
2) Centralized Online
3) Distributed Online

In the offline setting, everything required to solve the problem is known. This includes the data items requested, the path of each mobile client, their speed, and the transmission rate available at each BS. An example of this setting is a planned rescue mission. In this type of setting, all scheduling may be done before the mission starts.

In the centralized online setting, nothing about a job is known until it appears in the geographical region; then everything is known unless something unpredictable happens. An example of this type of setting is the delivery of data to buses in a city. Buses may be brought into service or removed from service, but their routes are predetermined. A central server may be made aware of the buses in service and schedule the delivery of information. In this setting we can reserve timeslots from a BS on the path of the mobile client.

In the distributed online setting nothing can be predicted, so every BS is in charge of its own timeslots; no reservations are allowed. This is a realistic model for cases when missions arrive spontaneously, such as from a fire alarm, and paths are planned dynamically. In these cases BSs must manage requests as they arrive.

We propose two overarching system architectures for generating the delivery schedules as described below.

In the centralized system a single server receives information updates from all locations and acts as the scheduler. This server is assumed to be powerful, able for example to solve IP (integer programming) problems. We use this server to solve the problem near-optimally by IP or offline algorithms. However, in practice unpredictable things may happen which degrade performance. For example, bandwidth may change, which does not fit directly in either the IP or the offline algorithms. Or in another case, new jobs appear from time to time, so frequent information updates will require that the centralized scheduling algorithms be run frequently, which may involve unacceptable system overhead. To address the limitation, we provide a distributed solution.

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

Although the performance of online distributed algorithms typically will be inferior to that of centralized algorithms, we will see in Section V that their performance can be quite good.

Table I. Comparison of System Architectures

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Centralized</td>
</tr>
<tr>
<td>Low</td>
<td>Distributed</td>
</tr>
</tbody>
</table>

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

IV. PROBLEM MODEL AND ALGORITHMS

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

A. Problem Models

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

Let \( r_{jk} \) and \( d_{jk} \) be the release time and deadline, respectively, for job \( j \) from the point of view of machine \( k \). Each job corresponds without loss of generality to a client traversing a route, seeking to obtain one data item. A client desiring multiple data items is assumed to be represented either by a client seeking a single complex item or as multiple clients seeking individual items while traveling the same route. In this paper, we do not consider potential difficulties involved in making this assumption.

\( w_{jk} \) is the profit to finish job \( j \) on machine \( k \). Rather than interpreting it as a machine-dependent profit, we use \( w_{jk} \) as a way to express “soft” deadlines. If a job has a soft deadline, we allow it to run late for partial profit. More accurately, this \( w_{jk} \) should be modeled as \( w_{jk} t \) because the profit varies over time. For simplicity, however, we assume this profit stays constant within the contact time window on each machine, and so it suffices to parameterize \( w \) by \( j, k \).

\( p_{jk} \) is the processing (or download) time for client \( j \) on machine \( k \), which, in some settings, may be based on a job size and a machine speed. Let \( S_{jk} \) indicate the starting time of job \( j \) on machine \( k \), if this assignment is chosen. In this case,
we must have \( r_{jk} \leq S_{jk} \) and \( S_{jk} + p_{jk} \leq d_{jk} \). A job instance is not a job but a potential assignment of a job, i.e., a feasible interval of size exactly \( p_{jk} \) on some machine \( k \) in which job \( j \) could be run. As noted above, a scheduling problem can be construed as an interval selection problem by replacing a job/machine pair, and its release time, deadline, and processing time, with the set of all possible corresponding job instances. Some of the algorithms, including the Two Phase [10], operate by considering job instances in order of increasing end time.

Machines have \( C \) channels on which to communicate with jobs, for some small constant \( C \). Each machine can communicate with at most one job at a time, per channel. Note that the channels of a machine can be thought of as \( C \) co-located single-channel machines. For practical reasons, we do not allow jobs to be paused and restarted, or to switch channels.

**B. IP Formulations**

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

Time-indexed formulations (see Dyer [14]) involve 0/1 decision variables for each possible assignment. This formulation involves only one set of decision variables, but they have three indices: job, machine, and timestep.

We define a variable \( x_{jks} \) for each job instance \( \{s, s + p_{jk}\} \) of \( J_j \) on \( M_k \). The objective is again to maximize the sum of weights of scheduled jobs. The Time-indexed Formulation is shown as follows:

\[
\max \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{s=r_{jk}}^{d_{jk}-p_{jk}} w_{jk} \cdot x_{jks} \\
\text{s.t.} \\
\sum_{j=1}^{n} \sum_{s=r_{jk}}^{d_{jk}-p_{jk}+1} x_{jku} \leq 1 \quad \forall k, s \\
\sum_{k=1}^{m} \sum_{s=r_{jk}}^{d_{jk}-p_{jk}} x_{jks} \leq 1 \quad \forall j \\
x_{jks} \in \{0, 1\}
\]

The first set of constraints prevents multiple jobs from being scheduled simultaneously on any single machine; the second set prevents any job from being scheduled more than once.

**C. Algorithms and Techniques for General Cases**

We this subsection begin by discussing approximation algorithms for general cases.

**D. Offline Algorithms**

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

We use the Two Phase algorithm from [10] as one of our offline algorithms. In the first phase, it pushes job instances in order of non-decreasing right endings onto a stack, if they have great enough weight relative to conflicting jobs on the stack; in the second phase, it pops job instances from the stack and places them in a nonoverlapping schedule. When a job enters the stack, it is pushed with the (strictly positive) difference of its profit and the profit of the overlapping jobs lower on the stack, with the effect that the total weight of the stack equals the weight of the schedule formed in the second phase. This algorithm provides a 2-approximation guarantee. See [10] for details of this algorithm.

Another combinatorial algorithm called Admission (see [9]) provides a \( 3 + 2\sqrt{2} \)-approximation. In this algorithm, jobs are considered in the order of non-decreasing endings. To apply it to \( m \) machines, we call the Admission algorithm \( m \) times (\( m \)-Admission), machine by machine. The \( 3 + 2\sqrt{2} \) ratio still holds in this situation [9]. We evaluate this algorithm as well as the Two Phase algorithm because it may be extended to work in the centralized online case.

**E. Centralized Online Algorithm**

As mentioned above, Admission works in real time order, so other than applying it machine by machine, it is easy to extend the algorithm to work on machines in parallel. We call the extended algorithm Global Admission (see Algorithm 1), in which we schedule the earliest finishing job among all the machines at each step.

**Algorithm 1 Global Admission**

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

The following criterion is used in \( m \)-Admission to decide whether a job should replace existing jobs it conflicts with. Let \( W \) be the total weight of all scheduled jobs overlapping with the current job \( j \). We accept job \( j \) if \( w_{jk} > W \cdot \beta \), for some constant \( \beta \).

The motivation to modify this criterion is illustrated in Figure 3. Two situations are shown, both with overlapping job instances of size 2 and 4. The job instance of size 2 is considered first in both cases because it ends first and we take jobs in order of increasing end time. If it is canceled for the job instance of size 4, the non-overlapping portion (if any) of its timeslots is wasted. So in case a one timeslot is
wasted if we cancel the first job instance, while in case b both timeslots of the first job are reused. As a result, we should expect higher weight on the second job in case a than in case b in order to replace the first job. However, a constant $\beta$ does not differentiate between the two cases.

![Fig. 3: Two cases of conflicting jobs.](image)

More generally, if the new job $j$ conflicts with a set of jobs $C_j$, let $\ell_j$ be the distance between the right endpoints of job $j$ and the rightmost job in $C_j$. (Note that all jobs conflicting with $j$ must end earlier than it.) And let $L_j$ be the span of $C_j$. We modify the criterion so that we accept job $j$ if $w_jk > W \cdot (1 + \ell_j/L_j)$. With this criterion, the size-4 job will be less likely to replace the size-2 job in case a than in case b. We tested our new criterion and found that it outperforms the old one. Due to space limitations, we do not include the test results in this paper. Unless otherwise mentioned, we use the new criterion with the Admission-related algorithms in our experiments.

We apply Global Admission iteratively to realize our Centralized Online algorithm (see Algorithm 2). We use Global Admission rather than Two Phase because it schedules jobs in real time. Two Phase is not suitable for real-time use because it does not begin scheduling jobs until after iterating through all jobs.

**Algorithm 2 Centralized Online**

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

The main idea of this algorithm is to reserve timeslots for a job on the machine on its path. The reservation is not guaranteed until the job starts executing on that machine, however. Until then, the reservation is subject to cancelation or modification as new jobs arrive. It is natural to view this as an incremental offline problem in which we run an offline algorithm at the moment when new jobs arrive.

For a highly dynamic system, it may become burdensome to run Global Admission too often.

**F. Distributed Online Algorithm**

In the Distributed Online setting, no reservations are allowed; each job requests timeslots from the machines within its communication range. Each time a machine receives a new job request $J_r$, it adds the job to its candidate list $I$. If no job is currently running, the machine schedules an available job $J_i$ maximizing $w_i/p_i$; if some job $J_r$ is currently running, we kill $J_r$ and schedule $J_i$ with the largest $o_i = w_i - w_r \cdot (1 + \frac{\ell_j}{p_r})$ (if this value $o_i$ is positive). Once a job is scheduled, it cancels its requests to other machines; once a job is killed, it can again request other machines.

**Algorithm 3 Distributed Online**

1: // occupied will be the last occupied timeslot  
2: // $I$ will be the set of unscheduled jobs  
3: occupied $\leftarrow 0$  
4: $I \leftarrow \emptyset$  
5: for each moment $t$ do  
6: given incoming job $J_j$, $I \leftarrow I \cup \{J_j\}$  
7: if occupied $\leq t$ then  
8: schedule $J_i \in I$ which has the highest $\frac{w_i}{p_i}$  
9: $I \leftarrow I \backslash \{J_i\}$  
10: occupied $\leftarrow t + p_i$  
11: else  
12: running job $J_r$  
13: $J_i \in I$ is job with the largest $o_i = w_i - w_r \cdot (1 + \frac{\ell_j}{p_r})$  
14: if $o_i > 0$ then  
15: replace job $J_r$ with $J_i$  
16: $I \leftarrow I \backslash \{J_r\} \cup \{J_i\}$  
17: occupied $\leftarrow t + p_i$  
18: end if  
19: end if  
20: remove from $I$ any jobs no longer schedulable  
21: end for

**V. Performance Evaluation**

In this section, we evaluate the algorithms, using both synthetic and real-world data.

**A. Mobility Pattern and Data Generation**

In our experiments, the movement of the cars follows one of two mobility patterns. For the first set of experiments, we use the Random Waypoint model described in [15], in which a node repeatedly selects a random destination in the simulation region and a speed from a range. The machines are deployed in a grid. In the second set of experiments, we extract mobility patterns from real mobility traces (obtained from UMass [2]) of buses encountering access points (APs) while following their routes.

In our tests, there is one job per trip. The job sizes are uniformly distributed in $[0, 50]$ (within a time horizon of size $t = 1000$). The job weights are chosen from a Zipf distribution, clipped with a minimum weight of 1 and a maximum weight of 10.

**B. Simulation with Random Waypoints**

To use the Random Waypoint model, for each problem instance, we simulate the trips of vehicles and record the time windows on every machine. Job sizes and weights are randomly generated as mentioned above.

In the first simulation, we fix the number of machines to 25 and vary the number of jobs. For each job count, we randomly generate 10 different problem instances and run the algorithms. We also solve the LP relaxation, which provides an upper
bound on the optimal solution value. We divide all other results by this upper bound, so the value shown is a lower bound of the approximation ratio.

When the system is not busy (see Figure 4), all the algorithms achieve close to the optimal. As the problem instance grows, however, the Distributed Online’s solution quality drops the fastest. The Centralized Online algorithm performs slightly worse than the two offline algorithms but does quite well.

![Fig. 4: Performance comparison, by instance size.](image)

To study the strengths of the different algorithms, we synthesized problem instances in which higher-weight jobs arrive over time. One motivation for this is applications in which more recent data is fresher and so given higher priority. Such instances are particularly hard for an online algorithm since it cannot know which current jobs to satisfy and which future jobs to wait for.

![Fig. 5: Evaluation, when](image)

Indeed, the Centralized Online algorithm’s performance drops faster than the offline algorithms’ on these instances (see Figure 5). Because it schedules current job without knowing the later jobs may have higher weight, although new jobs come with higher weight, it will keep on finishing the running jobs. The instances with increasing job weights emphasize this disadvantage. The Two Phase algorithm, which considers jobs in two directions, beats Admission, which schedules in real-time order. The disadvantage of scheduling in real-time order is that once it makes a decision, there is no chance to go back and revise it. One especially difficult decision in this problem is whether to preempt scheduled jobs. We have to make this decision more often when jobs appear with increasing weights because we only consider preemption when arriving jobs have higher weights than scheduled jobs. This explains why Admission performs slightly worse than Two Phase.

We use the same settings again in the third simulation, except this time varying the maximum job size (see Figure 6). As the max job size increases, the approximation factors of all algorithms drop at first and then begin rising again. The performance effects of increasing the max job size are complex. Our interpretation of the results is that at first, having larger jobs makes scheduling more difficult for the approximation algorithms. Over time, though, as the maximum job size (and hence the average job size) continues to grow, the problem becomes harder in the sense that the optimal solution value decreases, and hence the relative performance of the approximation algorithms actually improves.

![Fig. 6: Performance comparison, varying maximum job size.](image)

C. Simulation with Trace Files

We analyzed several bus/AP trace files obtained from UMass [2]. While these traces have useful mobility characteristics in terms of contact initiation and duration, there are not enough bus routes to simulate a heavily taxed system. Therefore, we replay several traces simultaneously, treating buses from different trace files as different buses, in order to obtain sufficiently many parallel jobs. We consider the m busiest BSs in these simulations.

We still limit the simulations to one job per bus. The jobs are generated randomly as described in the beginning of this section. As we can see (Figure 7), the algorithms tested achieve a very good approximation of the optimal. Note that the Centralized Online algorithm knows the arrival of jobs in future time windows.

VI. EXPLORING CASES WITH MACHINE-DEPENDENT WEIGHT

In this section, we start to explore the case in which jobs’ weights are changing over time. First we discuss several degradation functions and their meanings in practice. Then we test our algorithms with these weight degradation functions in problem settings similar to those in Section V.
A. Degradation Functions

Let \( w_j(t) = w_i \cdot f(t), f(t) \in [0, 1] \) indicate a degradation function. We evaluate three such functions (time \( t \in [0, 1] \) is normalized):

1) Linear decreasing: \( f(t) = 1 - t \)
2) Decreasing by \( \arccot: 0.5 - \arctan(10t - 5)/\pi \)
3) Linear increasing: \( f(t) = t \)

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

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

B. Algorithm Performance

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

In Figure 8(a), we can see that the Distributed Online runs very close to the Two Phase and that they both achieve near-optimal results, though the approximation ratios drop as we increase the number of jobs. In contrast, \( m \)-Admission and the Centralized Online algorithms performed poorly. The reason lies in the greedy choice Admission makes, scheduling an earliest-finish job, rather than basing the choice on job weights. The result could be that many jobs are scheduled on machines where their weights are low. The Two Phase algorithm is not affected by the varying weights because when it pushes job instance to the stack, it compares the weights of overlapping jobs. Finally, Distributed Online performs well because it picks the most valuable job competing for the current timeslots. The results of the second test (see Figure 8(b)) are very similar to those shown in Figure 8(a).

In the third test (see Figure 8(c)), the Two Phase performed well; the two algorithms based on Admission run slightly worse but still achieve better than the performance implied by their approximation guarantee. The Distributed Online algorithm did worse, with a solution quality of about 20% of the optimal. A possible explanation of this outcome is that when the weight increases, Distributed Online, which schedules the high-weight jobs as early as possible, works exactly the opposite way of what is expected, because those jobs with high weights are supposed to be scheduled as late as possible. (A job with large weight in the beginning is very likely to have a larger weight in the end.)

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

Table II summarizes the performance of these algorithms under various circumstances. We can see Two Phase is good for general use because it produces good or near-optimal results in all these situations. \( m \)-Admission performed equally well except for the time varying weights case. Because \( m \)-Admission is run iteratively, Centralized Online performed similarly to \( m \)-Admission; the online nature makes it hard to reach the offline algorithms’ performance, however. The Distributed Online performed well in all the cases except for the increasing weights setting, and it is the only one that naturally fits for the varying bandwidth case.

VII. CONCLUSION

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

ACKNOWLEDGEMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES