# 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 towards 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, and the in-transit "illumination" of simultaneous surface-to-air missiles. 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 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. Other examples may come from areas beyond mobile information access. Consider a scenario [1] in which a naval battle-group is defending itself against air attack and is equipped with surface-to-air-missiles, sensors, and target illuminators. When a missile is fired at the battle-group, it should be detected and illuminated by an illuminator at some point along its trajectory, in order that it be "homed-in on" and destroyed. As the missile flies through the air, it in general passes by different illuminators at different times, again yielding dependent time windows, this time for the "job" of illuminating a particular missile, at some point before it reaches its destination. With a limited number of illuminators, we naturally wish to illuminate as many of the missiles as possible.

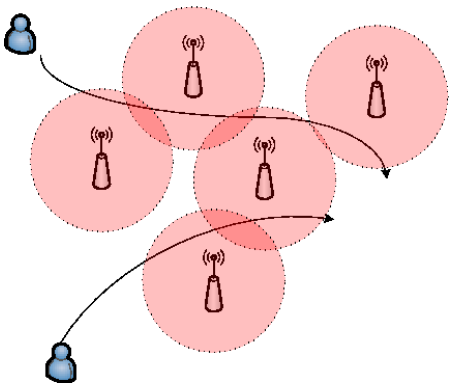


Fig. 1. Scenario: clients encountering BSs.

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 [2]. A valid assignment of job $j$ to machine $k$ is to dedicate $k$ (exclusively to $j$) over some interval $[s, s + p_j)$, with $[s, s + p_j) \subseteq [r_j, d_j)$. 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

978-1-4244-5113-5/09/$25.00 ©2009 IEEE

deadlines are machine-dependent, 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 [2], it is well studied with a few existing approximation algorithms. We will cover these algorithms in the Related Work section. In this paper, we adapt existing algorithms for related scheduling problems as well as 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 [3].

In addition to the general theoretical scheduling problems, we consider a realistic environment in which the available bandwidth at a BS varies over time (perhaps due to unforeseen requests for data) and the speed of the mobile clients varies. We adapt our algorithms using an estimated bandwidth and show the resilience of different 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. Finally, Section VI concludes the paper.

## II. Related Work

Our problem is about mobile data access [4]. In a mobile computing environment, users travel around carrying portable computing devices with wireless communication capability. As they move about, 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 unsharable. The challenge is how to allocate this resource to maximize the total profit.

A Content Distribution Network [5] 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 [6], 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 [2] for an introduction. Here we refer to some of the most relevant existing work. Lee et al. [1] 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 [1], 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. [1] 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.

In some versions of our problem, we assume additional structure beyond the core USP problem. Machines may be related in the following sense: job $j$ passing through a sequence of machines gives rise to $r_{jk}$ and $d_{jk}$ that increase monotonically (holding $j$ fixed).

When jobs are unit-size and machines are identical, the problem can be solved optimally by reduction to bipartite matching. More efficient algorithms are given by [7], however, with running times of $O(n \log n)$ for one machine and $O(mn^2)$ for $m$ machines. Many generalizations of this problem become NP-hard [7]: non-unit job sizes and job-dependent release times and deadlines, even for $m = 1$; $m > 1$ and identical release times and deadlines; $m > 1$ and identical release times but different deadlines; job sizes of $1, 3, q$ for some $q$; $m > 1$, arbitrary release times and deadlines, job sizes of $1, q$.

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 $\frac{(1+1/m)^m}{(1+1/m)^m-1)}$ (which approaches $e/(e-1)$ as the number of machines $m \to \infty$) for identical machines. The algorithm in both cases is called $m$-Greedy (in our notation), which applies an order-by-end-time greedy algorithm machine-by-machine. This algorithm can also be applied to the machine-dependent times setting.

A stack-based Two Phase algorithm with somewhat better performance is given by Berman and DasGupta [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 $t_2 - t_1$.

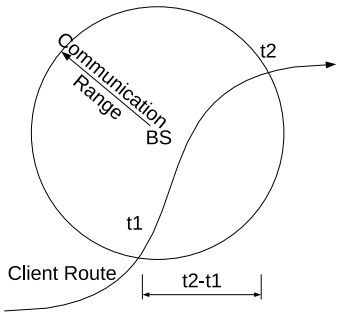



Fig. 2. Contact Time Windows

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. New buses may be placed into service or buses may be removed from service, but their routes are known. 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 their own timeslots; no reservations are allowed. This is a realistic setting in 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. This assumption may be unrealistic or constraining. In a highly dynamic environment, frequent information updates will require that the centralized scheduling algorithms be run frequently, which may involve unacceptable system overhead.

In a 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 information. Second, with each node using best-effort algorithms, such computation will be much faster involving 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.

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

TABLE I
Comparison of System Architectures

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 approximations for the general case of the problems followed by presenting optimal solutions for some special cases.

### A. Problem Models

Given are jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, each associated with a profit $w_j$, and machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. The clients roaming through the geographical region come within transmission range of some of the machines at different times. This

$$\max \sum_{j=1}^{n}\sum_{k=1}^{m} w_j \cdot y_{jk} \quad (1)$$

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

TABLE II
Start-time Formulation

$$\max \sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{s=r_{jk}}^{d_{jk}-p_{jk}} w_j \cdot x_{jks} \quad (2)$$

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

TABLE III
Time-indexed Formulation

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 $j$ when run on 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.

$p_{jk}$ 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 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. We do not allow jobs to be paused and restarted, nor to switch channels.

### B. IP Formulations

Our problem can be formulated as an integer program (IP), and although solving IPs is NP-hard in general, it is frequently possible in reasonable time for moderately sized instances. For larger instances, solving a linear programming (LP) relaxation in polynomial time can also provide useful upper bounds on solution quality.

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

*1) Start-time Formulation:* We extend the single-machine formulation of [14] to multiple machines, by introducing an additional summation index $k$. Following [14], we define the following variables:

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

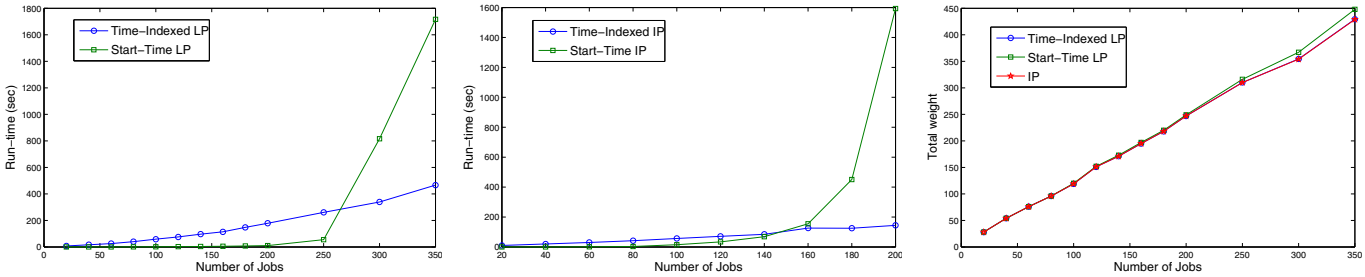
$$x_{ijk} = \begin{cases} 1 & \text{if job } i \text{ is performed before job } j \text{ on} \\ & \text{machine } k, \text{ or job } j \text{ is performed} \\ & \text{on machine } k \text{ and job } i \text{ is not,} \\ 0 & \text{otherwise} \end{cases}$$

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

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

*2) Time-indexed Formulation:* Time-indexed formulations (see Dyer [15]) involve 0/1 decision variables for each possible assignment. This formulation involves only one set of decision variables, but they have three indices: job, machine, and timestep.

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




(a) Time to solve the LP relaxations.

(b) Time to solve the IPs.

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

Fig. 3. Two Formulations

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.

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

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

TABLE IV
COMPLEXITY OF TWO FORMULATIONS

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

Figures 3(a) and 3(b) compare the solver running times of the two LP-relaxations and two IPs respectively. For small instances of the problem the Start-time LP formulation is solved faster as expected. For larger instances though solving Time-indexed LP is more efficient. However, the two LP-relaxations give different upper-bounds. The difference is significant for larger problem instances. In fact, [16] claims that the bounds provided by the solution to the LP-relaxation of a time-indexed formulation to be stronger than the bounds provided by the LP-relaxation of many other integer programming formulations. This leads to more robust branch-and-cut algorithms for time-indexed formulation. Figure 3(c) shows that the upper-bound for the Time-indexed LP almost coincides with the solution to the IP. The Start-time LP gives a higher upper-bound.

As a result, in practice we prefer using the Time-indexed formulation in our experiments because the running time increases linearly as we increase the problem size. In cases in which the Start-time formulation is faster, i.e., when $n^2m \ll nmt$, we choose it.

### C. Algorithms and Techniques for General Cases

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

*1) Offline Algorithm:* In offline settings, we can try to solve the IP problem. Although this may take too long for realistic problem instances, we can use these solutions to evaluate other algorithms. In fact, we provide solutions to the LP as a bound on the optimal solution. In practice, we typically find the LP and IP 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 Algorithm 1 for details of this algorithm.

Another combinatorial algorithm called Admission (see [9]) has 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.

*2) Centralized Online Algorithm:* As mentioned above, Admission works in real time order, so other than applying it machine by machine, it is easy to extend the algorithm to work on machines in parallel. We call the extended algorithm Global Admission (see Algorithm 2), in which we schedule the earliest finishing job among all the machines at each step.

The following criterion is used in $m$-Admission to decide whether a job should replace existing jobs it conflicts with.

**Algorithm 1** Two Phase Algorithm

$L \leftarrow$ the set of all job instances *(job, weight, beginning, ending)*
sort $L$ so the ending is non-decreasing
$S \leftarrow$ an empty stack
**for** each $(i, w, d, e)$ from $L$ **do**
  $v \leftarrow w - total(i, d) - TOTAL(d)$
  **if** $v > 0$ **then**
    push$((i, v, d, e), S)$
  **end if**
**end for**
**for** each $i$ **do**
  $done[i] \leftarrow$ false
**end for**
$occupied \leftarrow t$
**while** $S$ is not empty **do**
  $(i, v, d, e) \leftarrow$ pop($S$)
  **if** $done[i] =$ false and $e \leq occupied$ **then**
    add $(i, d, e)$ to solution
    $done[i] \leftarrow$ true, $occupied \leftarrow d$
  **end if**
**end while**

**Algorithm 2** Global Admission

$A \leftarrow \emptyset$
$I$ is the set of all job instances
**while** $I$ is not empty **do**
  let $J_j \in I$ be a job instance that terminates earliest
  $I \leftarrow I \setminus \{J_j\}$
  let $C_j$ be the set of jobs in $A$ overlapping with $J_j$
  let $W$ be the total weight of $C_j$
  **if** $W = 0$ or $w_j > W \cdot (1 + \frac{\ell_j}{L_j})$ **then**
    $A \leftarrow A \cup \{J_j\} \backslash C_j$
  **end if**
**end while**
return $A$

Let $W$ be the total weight of all scheduled jobs overlapping with the current job $j$. We accept job $j$ if $w_j > W \cdot \beta$, for some constant $\beta$.

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

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 $w_j > W \cdot (1 + \ell_j / L_j)$. With this criterion, the size-4 job will be less likely to replace the size-2 job in case *a* than in case *b*.

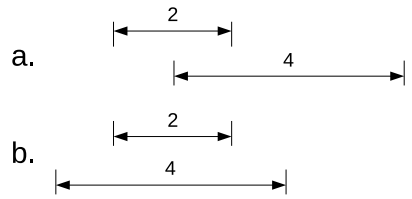



Fig. 4. Two cases of conflicting jobs.

We apply Global Admission iteratively to realize our Centralized Online algorithm (see Algorithm 3). 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 3** Centralized Online

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

*3) Distributed Online Algorithm:* In the Distributed Online setting, no reservations are allowed; each job requests timeslots from the machines within its communication range. Each time a machine receives a new job request $J_i$, 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 $o_i = w_i - w_r \cdot (1 + \frac{\ell_i}{p_r})$ (if this $o_i$ is positive). Once a job is scheduled, it cancels its requests to other machines; once a job is killed, it can again request other machines.

### *D. Optimal Algorithms for Special Cases*

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

**Algorithm 4** Distributed Online

```
// occupied = last occupied timeslot
// I = set of unscheduled jobs
occupied ← 0
I ← ∅
for each moment t do
    given incoming job J_j, I ← I ∪ {J_j}
    if occupied ≤ t then
        schedule J_i ∈ I which has the highest w_i / p_i
        I ← I\{J_i}, occupied ← t + p_i
    else
        running job J_r
        J_i ∈ I is job with the largest o_i = w_i − w_r · (1 + ℓ_i / p_r)
        if o_i > 0 then
            replace job J_r with J_i
            I ← I ∪ {J_r}\{J_i}, occupied ← t + p_i
        end if
    end if
    remove from I any jobs no longer schedulable
end for
```

*1) Unit Processing Time:* If each job has unit processing time, the problem can be reduced to a maximal matching problem in which we match jobs to timeslots. This maximal matching problem can be solved optimally by e.g. the Hungarian algorithm.

In one even more special case, where release time is zero ($r_{jk} = 0$) and each machine encounters the jobs in the same order, the problem can be solved by a more efficient algorithm in $O(n \log(n))$ time. We refer to this type monotonicity as **mono1**[1]. For this setting, we can adopt the optimal solution for the case without machine-dependent windows [17]. In that algorithm, we sort jobs by their deadline (the same order on all machines), then iteratively we put jobs in the earliest available slot. If there are no available slots, replace a scheduled job of minimum weight if less than the new job's weight.

*2) Arbitrary Processing Time with **mono1**:* If we generalize this problem with arbitrary but equal processing times on every machine, it is solvable in pseudo-polynomial time by applying the dynamic programming techniques of Rothkopf [18] and Lawler and Moore [19]. In the machine-dependent deadlines / release time zero case, different machines encounter jobs in the same order, and so we can directly apply the DP algorithms. Unfortunately, the running time and space are both $O(n(\sum p_j)^m)$, which quickly becomes impractical for moderate values of $m$. Neither of these algorithms applies to the setting with varying release times, however.

## V. Performance Evaluation

### A. Mobility Pattern and Data Generation

We use two different mobility patterns in our experiments. For the first set of experiments, we use the Random Waypoint model described in [20], in which a node 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 out of real mobility traces (from UMass [3]) of buses encountering access points (APs) while traveling on 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.

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

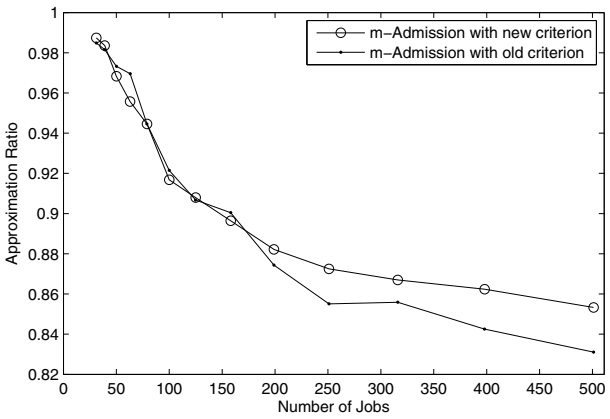



Fig. 5. Performance of the $m$-Admission variants.

### B. Simulation with Random Waypoints

In the first simulation, we fix the number of machines to be 25 and we increase the number of jobs. For each of $n$ jobs, we run 10 different random generated scenarios. 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 the lower bound of the approximation ratio. Before we conduct the simulation to compare the algorithms, we tested our new criterion mentioned in Section IV-C using the $m$-Admission algorithm. As the load increases in the system (see Figure 5), the new criterion performs better than the old one. Unless otherwise mentioned, we use the new criterion with the Admission related algorithms in our experiments.

When the system is not busy (see Figure 6), all the algorithms achieve close to the optimal. As the problem instance becomes larger, 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.

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 more fresh 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.

Indeed, the Centralized Online algorithm's performance

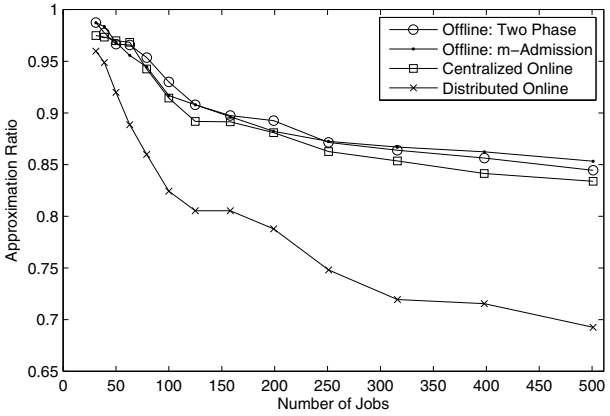



Fig. 6. Performance comparison, by instance size.

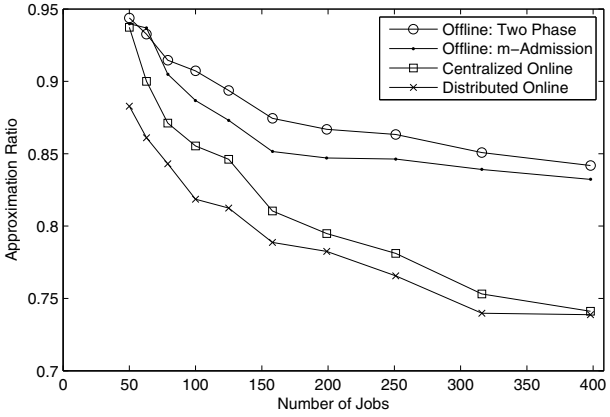



Fig. 7. Evaluation, when jobs appear with increasing weights.

drops faster than the offline algorithms' (see Figure 7). The Two Phase algorithm, which considers jobs in two directions, beats Admission, which schedules in real time order.

In the third simulation, we again use the same settings, except this time varying the max job size (see Figure 8). As the max job size increases, the approximation ratios 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 max job size (and hence the average job size) continues to grow, the problem becomes harder, i.e. the optimal solution value decreases, and hence the relative performance of the algorithms improves.

We also tested several special mobile routes:

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

Although problem difficulty varies *between* these three settings, within each setting we generally see similar patterns

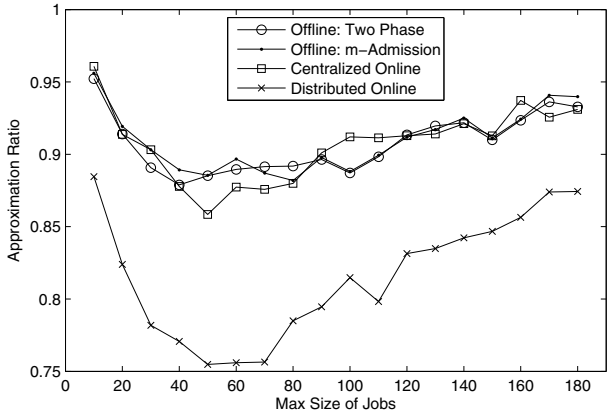



Fig. 8. Performance comparison, varying maximum job size.

among the algorithms' performance. The results of Figure 9(a) are very similar to those in the general setting (see Figure 6); because the competition starts when jobs are still relatively away from each other, many jobs get scheduled before they reach the center. Figure 9(b) shows a fast drop in the two online algorithms' performance because of the severe competition at the start, when jobs are very close to each other. Figure 9(c) shows some interesting behavior. $m$-Admission seems to consistently beat the Two Phase, and Centralized Online performs worse than Distributed Online when there are few jobs. A possible explanation for the latter effect could be the fact that the centralized algorithm does not allow cancelation of the executing jobs when Global Admission is run.

### C. Simulation with Trace Files

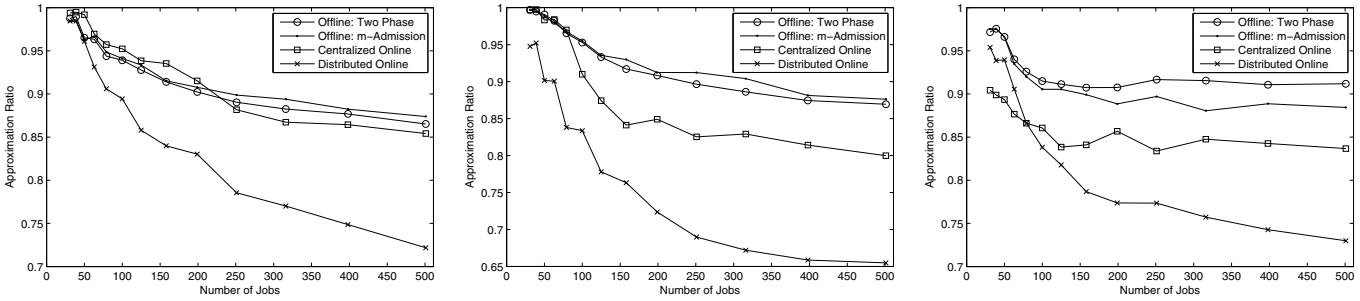
We analyzed several bus/AP trace files obtained from UMass [3]. While these traces have useful mobility characteristics in terms of contact initiation and duration, there are not sufficiently many bus routes to simulate a heavily taxed system. Therefore, we replay several traces simultaneously and treat buses from different trace files as different buses, in order to obtain sufficiently many parallel jobs. We consider the $m$ busiest BSs in these simulations.

A histogram of the window sizes found in this dataset is shown in Figure 10. Based on this data, we chose a max job size of 50, since larger jobs would not be schedulable anywhere.

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 11), 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.

### D. Varying Bandwidth

More realistically, network conditions can vary, so the effective transmission rates at BSs may change over time. In this case, the processing time $p_{jk}$ not only depends on the




(a) Converging towards a central location. (b) Diverging from a central location. (c) Traversing the same route.

Fig. 9. Special traffic patterns.

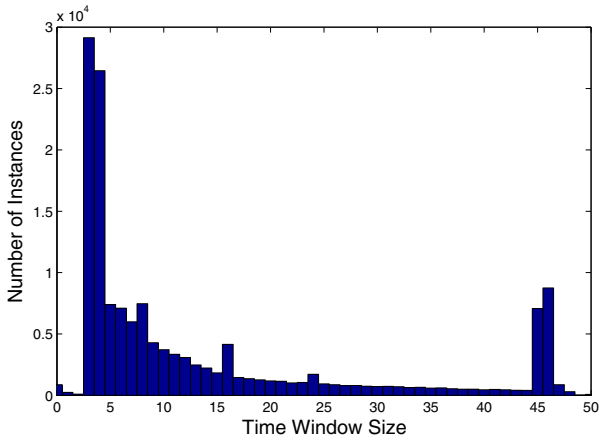



Fig. 10. Histogram of time window size

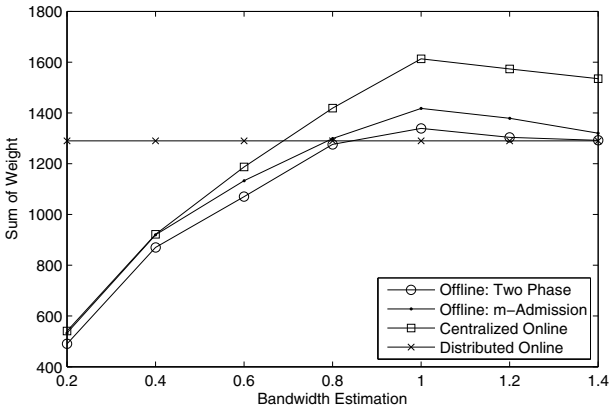



Fig. 12. Performance comparison, increasing the estimated bandwidth.

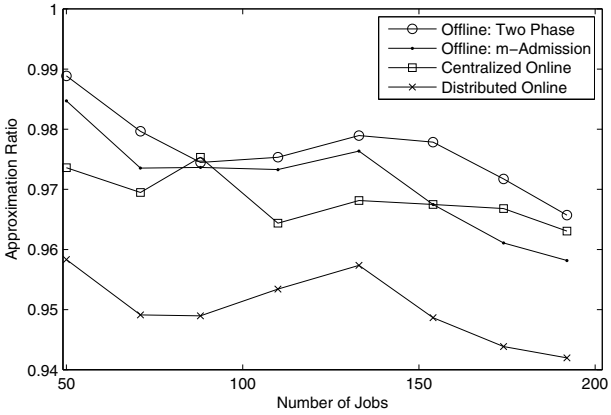



Fig. 11. Performance comparison, increasing the number of jobs.

job and machine pair, but also depends on the time. Because integrating this generalization into the IP formulations would yield intractably large programs, we do not provide an optimal bound for comparison.

The difficulty is that we cannot realistically know how the network environment will change, and recall that except for the Distributed Online algorithm, the algorithms assume a fixed bandwidth on all machines. We cannot expect to know in advance about such bandwidth changes; since the offline algorithms make decisions in advance, we need to rely on bandwidth estimates; the centralized online algorithm will also use these estimates. Unfortunately, if these estimates are over-confident, then the schedule produced may not actually be feasible; if the estimates are over-pessimistic, then this means wasting bandwidth we could have used. The Distributed Online algorithm uses a real-time rate estimate for current bandwidth, but it may also experience the problem of producing infeasible schedules since bandwidth may change as a job is executed.

To investigate this situation, we conducted the following experiment: the bandwidth of each machine changes over time, following a normal distribution with expectation $0.5$ and variance $0.2$, with the distribution clipped at $[0.1, 1]$. As the bandwidth changes, the processing time changes accordingly. We again use the UMass traces to generate time windows, with other parameters set as in the previous tests. In order for our algorithms to achieve good performance, we increase the estimated bandwidth to see the impact on our solutions and to find the best bandwidth estimation. The results shown in Figure 12 plot the performance of the different algorithms.

From Figure 12, first of all we can see that the Distributed Online algorithm gets the same results because it uses its own real time estimation. When the rates are underestimated, the three other algorithms performed very poorly. This is because

we waste many timeslots by assigning them to some jobs that actually need fewer timeslots. As the estimated rate increases, the offline and centralized algorithms begin to outperform the distributed online algorithm. When we overestimate the rates, however, the total weight drops. Many scheduled jobs do not complete since they require more time than allocated.

Interestingly, the centralized online beats the $m$-Admission, which in turn beats the Two Phase. In the fixed bandwidth situation, we expect the opposite; in the case of varying bandwidth, however, scheduling jobs too tightly has a negative effect; leaving some extra space between jobs makes the schedule more robust, providing some leeway to accommodate varying rates. The distributed algorithm naturally fits this situation without any estimation on rate required while still performing well.

## VI. 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.

Several interesting open problems are raised by our work:

1) There are various ways in which a job's time windows on different machines could be related.
2) The offline algorithms guarantee the performance in the worst case, however, we may expect a better approximation in general cases.
3) A BS may have information about its neighbors, or several close BSs could work jointly in a group to improve the Distributed Online algorithm.
4) The Centralized Online algorithm works iteratively, each time rescheduling most jobs including the ones having reservations. Overhead could be reduced by an incremental algorithm that attempts to limit the number of revisions made to the existing schedule.
5) For the varying bandwidth case, a time index $s$ could be added to the to processing time $p_{jk}$ so that we can try to solve it offline. Although making this modification in a naive way would significantly enlarge the formulations, perhaps a more tractable modification is possible.

## Acknowledgments

This research was sponsored by US Army Research laboratory and the UK 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 authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## References

[1] Y. Lee and H. Sherali, "Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battlegroup problem," *Annals of Operations Research*, vol. 50, 1994.
[2] P. Brucker, *Scheduling algorithms*. Springer, 2004.
[3] J. Burgess, B. N. Levine, R. Mahajan, J. Zahorjan, A. Balasubramanian, A. Venkataramani, Y. Zhou, B. Croft, N. Banerjee, M. Corner, and D. Towsley, "CRAWDAD data set umass/diesel (v. 2008-09-14)," Downloaded from http://crawdad.cs.dartmouth.edu/umass/diesel, Sep. 2008.
[4] J. Jing, A. S. Helal, and A. Elmagarmid, "Client-server computing in mobile environments," *ACM Computing Surveys*, vol. 31, no. 2, 1999.
[5] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 169–182, 2001.
[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 56–67, 2000.
[7] B. Simons and M. Warmuth, "A fast algorithm for multiprocessor scheduling of unit-length jobs," *SIAM Journal of Computing*, vol. 18, no. 4, August 1989.
[8] R. He, "Parallel machine scheduling problem with time windows: A constraint programming and tabu search hybrid approach," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, 2005.
[9] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, "Approximating the throughput of multiple machines in real-time scheduling," *SIAM Journal of Computing*, vol. 31, no. 2, 2001.
[10] P. Berman and B. DasGupta, "Multi-phase algorithms for throughput maximization for real-time scheduling," *Journal of Combinatorial Optimization*, vol. 4, no. 3, 2000.
[11] G. Koren and D. Shasha, "D over; an optimal on-line scheduling algorithm for overloaded real-time systems," in *Real-Time Systems Symposium, 1992*, 1992, pp. 290–299.
[12] G. Woeginger, "On-line scheduling of jobs with fixed start and end times," *Theoretical Computer Science*, vol. 130, no. 1, pp. 5–16, 1994.
[13] J. Ding, T. Ebenlendr, J. Sgall, and G. Zhang, "Online scheduling of equal-length jobs on parallel machines," *Lecture Notes in Computer Science*, vol. 4698, p. 427, 2007.
[14] N. G. Hall and M. J. Magazine, "Maximizing the value of a space mission," *European journal of operational research*, vol. 78, no. 2, pp. 224–241, 1994.
[15] M. E. Dyer and L. A. Wolsey, "Formulating the single machine sequencing problem with release dates as a mixed integer program," *Discrete Applied Mathematics*, vol. 26, no. 2-3, pp. 255–270, 1990.
[16] J. M. van den Akker, C. A. J. Hurkens, and M. W. P. Savelsbergh, "Time-Indexed Formulations for Machine Scheduling Problems: Column Generation," *INFORMS Journal on Computing*, vol. 12, no. 2, pp. 111–124, 2000.
[17] P. Brucker and S. A. Kravchenko, "Preemption can make parallel machine scheduling problems hard," *Osnabrucker Schriften zur Mathematik, Reihe P*, 1999.
[18] M. H. Rothkopf, "Scheduling Independent Tasks on Parallel Processors," *Management Science*, vol. 12, no. 5, pp. 437–447, 1966.
[19] E. L. Lawler and J. M. Moore, "A functional equation and its application to resource allocation and sequencing problems," *Management Science*, vol. 16, no. 1, pp. 77–84, 1969.
[20] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. ACM New York, NY, USA, 1998, pp. 85–97.