Technical Report: Optimal Parallel Scheduling under Concave Speedup Functions

Chengzhang Li[†], Peizhong Ju[‡], Atilla Eryilmaz[†], Ness B. Shroff[†]
[†]Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH, USA
[‡]Department of Computer Science, University of Kentucky, Lexington, KY, USA

ABSTRACT

Efficiently scheduling parallel computing resources across multiple concurrent jobs is a core challenge in modern cloud and edge computing systems that support AI-driven applications. Allocating more resources to a job accelerates its completion, but with diminishing returns. Prior work (heSRPT) has solved this problem for certain speedup functions with an exponential form, providing a closed-form solution. However, the general case with arbitrary concave speedup functions—which more accurately capture real-world workloads—has remained open.

In this paper, we solve this open problem by developing optimal scheduling algorithms for parallel jobs under general concave speedup functions. We first discover a fundamental and broadly-applicable rule for optimal parallel scheduling, namely the *Consistent Derivative Ratio (CDR) Rule*, which states that the ratio of the derivatives of the speedup functions across active jobs remains constant over time. To efficiently compute the optimal allocations that satisfy the CDR Rule, we propose the *General Water-Filling (GWF)* method, a more general version of classical water-filling in wireless communications. Combining these insights, we design the *SmartFill* Algorithm to solve the general scheduling problem. Unlike heSRPT, which always allocates resources to all active jobs, SmartFill selectively determines which jobs should receive resources and how much they should be allocated. For a broad class of so-called *regular* speedup functions, SmartFill yields closed-form optimal solutions, while for non-regular functions it efficiently computes the optimum numerically. Simulation results show that SmartFill can outperform heSRPT across a wide range of concave speedup functions.

KEYWORDS

Parallel scheduling, concave speedup function, resource allocation

ACM Reference Format:
Chengzhang Li[†], Peizhong Ju[‡], Atilla Eryilmaz[†], Ness B. Shroff[†] . 2026. Technical Report: Optimal Parallel Scheduling under Concave Speedup Functions. In *Proceedings of the 27th International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (ACM MobiHoc '26).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACM MobiHoc '26, November 23–26, 2026, Tokyo, Japan
© 2026 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Parallel computing has become indispensable in modern computing systems [1]. At the cloud scale, data centers such as Microsoft Azure [2], Amazon Web Services (AWS) [3], and Google Cloud [4] operate with tens of thousands of GPUs per site. Even at the edge, powerful devices such as the NVIDIA A100 GPU provide up to 6,912 CUDA cores that can run in parallel [5]. Modern AI workloads are explicitly designed to exploit this massive parallelism. For example, training deep neural networks such as Transformers [6] and large language models [7] leverages parallel GPUs to significantly accelerate training [8, 9], while GPU-based inference of AI models reduces latency in many applications [10, 11].

When multiple jobs compete for parallel-computing resources, scheduling becomes a critical task [12]. A common objective is to minimize job completion times or related performance metrics [13]. The efficiency of any scheduling policy depends heavily on the *speedup function*, which describes how a job's processing speed increases as more parallel resources are allocated. If the speedup function is linear, the classic SRPT algorithm [14] is optimal. In practice, however, speedup functions are typically concave, reflecting diminishing returns as additional resources are assigned [15, 16]. This makes the scheduling problem substantially more challenging.

Berg *et al.* [17] studied this problem under a special family of speedup functions $s(\theta) = \theta^p$ for $0 < p < 1$. They derived elegant theoretical results and a closed-form optimal scheduling policy in this setting. Although this family of speedup functions reflects some empirical measurements of real-world applications (see Fig. 2 in [17]), it does not capture the general behavior of concave functions, which limits the broader applicability of the results. The existing literature [16] identified the problem of scheduling under general concave speedup functions as "an entirely open problem." More recently, Ghanbarian *et al.* [15] studied a scheduling problem with general concave speedup functions under a different system model (e.g., assuming Poisson job arrivals and blocking jobs when no resource is available), and provided an asymptotically optimal solution in that setting. However, due to differences in system models, their asymptotically optimal solution is essentially a probabilistic greedy scheme, and it does not solve the open problem identified in [16].

In this paper, we solve this open problem by developing optimal scheduling algorithms for parallel jobs with *general concave speedup functions*. Our system model and objective are identical to those in [17], except that we allow arbitrary concave speedup functions instead of restricting to $s(\theta) = \theta^p$. This generalization substantially broadens applicability to diverse workloads and computing platforms.

We make two key theoretical contributions. First, we establish the *Consistent Derivative Ratio (CDR) Rule*, which states that under

any optimal policy, the ratio of the derivatives of the speedup functions across active jobs remains constant over time. This property dramatically reduces the search space, since any optimal schedule must satisfy it. We prove the CDR Rule by contradiction: if the derivative ratio were not constant, a small resource reallocation between two jobs would strictly reduce the objective function, violating optimality.

Second, we propose the *General Water-Filling (GWF)* algorithm, which efficiently computes allocations that satisfy the CDR Rule. GWF generalizes the classical water-filling algorithm in wireless communications [18]: while the classical setting assumes identical channel widths, GWF accommodates "bottles" of different widths and shapes. The bottles are defined based on so-called *auxiliary functions*, and we provide a method for constructing them. For a broad family of *regular speedup functions*, including $s(\theta) = \theta^p$, GWF produces closed-form optimal solutions. For non-regular concave functions, it produces efficient numerical optimal solutions. In contrast to existing generalizations of the water-filling algorithm [19, 20], which focus on static optimization settings, our proposed GWF is tailored to a dynamic resource allocation problem and introduces new algorithmic designs and insights, including the auxiliary function $g(h)$ and the notion of regular" speedup functions.

Building on these results, we design the *SmartFill* algorithm, which integrates the CDR Rule and GWF into a complete solution. SmartFill can solve the parallel scheduling problem optimally under general concave speedup functions. Unlike heSRPT, which allocates some resource to every active job, SmartFill intelligently determines which jobs should receive resources and how much should be allocated. Through numerical evaluation, we show that SmartFill consistently outperforms heSRPT across various concave speedup functions.

Finally, we discuss the broader applicability of the results in this paper. The CDR Rule is in fact a very general principle in parallel scheduling. It continues to hold under more general settings, including scenarios with heterogeneous speedup functions across jobs and time-varying total system resources, thereby significantly reducing the search space of possible schedules. However, the GWF and SmartFill algorithms do not directly extend to these settings. We identify this as an important open problem for future research.

2 SYSTEM MODEL AND PROBLEM STATEMENT

We consider the *exact same* system model as in [17], with the only difference being the assumption on the speedup function. Specifically, we assume the speedup function is a general concave function, rather than an exponential one. While some notations used here differ slightly from those in [17], they are mathematically equivalent.

We consider a system with M parallelizable jobs, each available at time $t = 0$. Let x_i denote the size of job i for $i = 1, 2, \ldots, M$. Without loss of generality, we assume that job sizes follow a non-increasing order:

$$x_1 \geq x_2 \geq \cdots \geq x_M.$$

The system includes a divisible server that allocates computation resource among the jobs at any time $t > 0$. Let B denote the total resource of the server, and let $\theta_i(t)$ denote the amount of resource allocated to job i at time t. The allocations must satisfy the constraint:

$$\sum_{i=1}^{M} \theta_i(t) \leq B, \quad \forall t > 0. \tag{1}$$

We assume that $\theta_i(t)$ is *right-continuous* w.r.t. t. This assumption rules out discontinuities at isolated points[1].

We assume all jobs follow the same speedup function $s(\theta)$ defined in $\theta \in [0, B]$, which denotes the service rate for a job when allocated with resource θ. Let $s'(\theta)$ denote the derivative of $s(\theta)$. We assume that the speedup function satisfies:

- $s(0) = 0$,
- $s(\theta)$ is strictly increasing w.r.t. θ,
- $s(\theta)$ is continuous and differentiable w.r.t. θ,
- $s(\theta)$ is strictly concave w.r.t. θ,
- $s'(\theta)$ is continuous w.r.t. θ.

Since $s(\theta)$ is strictly increasing and strictly concave, we have $s'(\theta) > 0$ and $s'(\theta)$ is strictly decreasing for $\theta \in [0, B]$.

Note that in [17], the authors considered a special family of speedup functions, $s(\theta) = \theta^p$, for $0 < p < 1$. While this family is simple to analyze, it fails to capture the behavior of most concave functions. A key limitation is that these functions all satisfy $s'(0) = \infty$. This property is unrealistic in many real-world applications, where allocating a small amount of resource to a job provides only limited benefit rather than an infinite one. Moreover, this property fundamentally alters the structure of the optimal scheduler, as we will further discuss in Section 4.

Let $Q_i(t_1, t_2)$ denote the amount completed service for job i between time period $[t_1, t_2]$. We have

$$Q_i(t_1, t_2) = \int_{t_1}^{t_2} s(\theta_i(t))\, dt, \quad \text{for } i = 1, 2, \cdots, M \text{ and } \forall t_1 < t_2. \tag{2}$$

Let T_i denote the completion time of job i. For each job $i = 1, 2, \cdots, M$, we have:

$$\theta_i(t) = 0, \quad \text{for } t > T_i. \tag{3}$$

and

$$Q_i(0, T_i) = x_i, \quad \text{for } i = 1, 2, \cdots, M. \tag{4}$$

Let J denote the system performance metric, which is a weighted sum of job completion times:

$$J = \sum_{i=1}^{M} w_i T_i. \tag{5}$$

Here w_i is the weight for job i. Following the setting in [17], we assume that the weights follow a non-decreasing order:

$$w_1 \leq w_2 \leq \cdots \leq w_M.$$

This ordering implies that shorter jobs are given higher priority through larger weights. The choice of w_i depends on the performance metric of interest. For example, when $w_1 = w_2 = \cdots = w_M = 1$, J corresponds to the mean flow completion time [13]. Alternatively, when $w_i = \frac{1}{x_i}$, J represents the mean slowdown of the system [21].

[1] Alternatively, assuming left-continuity yields the same theoretical results throughout the paper.

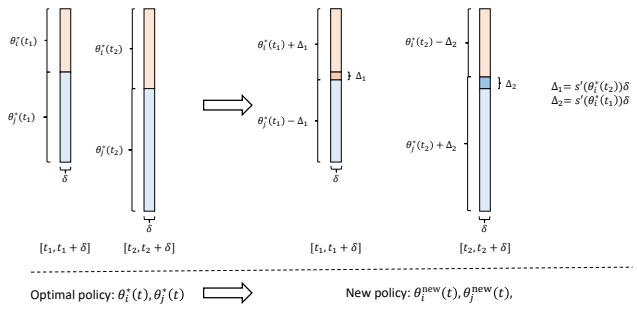

Figure 1: Idea of the proof of Theorem 1.

The objective of this paper is to find the optimal allocation $\theta_i(t)$ for each job i and time t, so as to minimize J, given B, x_i, $s(\theta)$, and w_i's. Formally, we aim to solve the following optimization problem:

$$\text{OPT:} \quad \min_{\theta_i(t)} \quad J = \sum_{i=1}^{M} w_i T_i,$$
$$\text{subject to} \quad \text{Constraints (1), (2), (3), (4).}$$

The remainder of this paper is devoted to solving OPT. In Section 3, we present our first key result: a structural property of the optimal scheduling policy, the *Consistent Derivative Ratio (CDR) Rule*, which significantly reduces the search space of OPT. Section 4 introduces our second key result: the *General Water-Filling (GWF)* algorithm, which efficiently computes allocations based on the CDR Rule. Finally, in Section 5, we combine these key results to develop the *SmartFill* algorithm, which provides a complete solution to OPT.

3 CONSISTENT DERIVATIVE RATIO RULE

In this section, we present the main result of this paper—the *Constant Derivative Ratio (CDR) Rule.*

Let $\theta_i^*(t)$ denote an optimal scheduling policy to OPT. The following theorem establishes that, under the optimal solution, the ratio of the derivatives of the speedup functions between any two jobs remains constant over time, as long as both jobs have a positive service rate.

Theorem 1. ***(CDR Rule)**: For any pair of jobs i and j, there exists a constant $c_{i,j}$ such that*

$$\frac{s'(\theta_i^*(t))}{s'(\theta_j^*(t))} = c_{i,j}, \tag{6}$$

for all t such that $\theta_i^*(t) > 0$ and $\theta_j^*(t) > 0$.

Proof. We prove Theorem 1 by contradiction.

Assume the theorem is false. Then, there exist two time instances t_1 and t_2 such that

$$\frac{s'(\theta_i^*(t_1))}{s'(\theta_j^*(t_1))} > \frac{s'(\theta_i^*(t_2))}{s'(\theta_j^*(t_2))},$$

with $\theta_i^*(t_1) > 0$, $\theta_j^*(t_1) > 0$, $\theta_i^*(t_2) > 0$, and $\theta_j^*(t_2) > 0$.

Define $h_{i,j} = s'(\theta_i^*(t_1))s'(\theta_j^*(t_2)) - s'(\theta_i^*(t_2))s'(\theta_j^*(t_1))$. By the assumption, we have $h_{i,j} > 0$.

The idea of the proof is shown in Fig. 1. We will construct a new scheduling solution $\theta_i^{\text{new}}(t)$ and $\theta_j^{\text{new}}(t)$ that decreases the objective value, while keeping all other scheduling decisions unchanged.

We construct $\theta_i^{\text{new}}(t)$ and $\theta_j^{\text{new}}(t)$ as follows. In the interval $[t_1, t_1+\delta]$, we increase the service rate for i by a small amount $\Delta_1 = s'(\theta_i^*(t_2))\delta$ and decrease the service rate for j by the same amount:

$$\theta_i^{\text{new}}(t) = \theta_i^*(t) + \Delta_1,\ \theta_j^{\text{new}}(t) = \theta_j^*(t) - \Delta_1, \quad \text{for } t \in [t_1, t_1+\delta].$$

Let $Q^*(\cdot)$ and $Q^{\text{new}}(\cdot)$ denote the amount of service under the original and modified schedules, respectively. Consider the amount of service for job i in time interval $[t_1, t_1+\delta]$. We have

$$Q_i^{\text{new}}(t_1, t_1+\delta) - Q_i^*(t_1, t_1+\delta) = \int_{t_1}^{t_1+\delta} \big(s(\theta_i^{\text{new}}(t)) - s(\theta_i^*(t))\big)dt$$
$$= \int_{t_1}^{t_1+\delta} \big(s(\theta_i^*(t)+\Delta_1) - s(\theta_i^*(t))\big)dt$$
$$= \int_{t_1}^{t_1+\delta} \big(s'(\theta_i^*(t))\Delta_1 + o(\Delta_1^2)\big)dt$$
$$= s'(\theta_i^*(t))\Delta_1\delta + o(\delta^3) = s'(\theta_i^*(t_1))s'(\theta_i^*(t_2))\delta^2 + o(\delta^3)$$

Then we have:

$$Q_i^{\text{new}}(t_1, t_1+\delta) = Q_i^*(t_1, t_1+\delta) + s'(\theta_i^*(t_1))s'(\theta_i^*(t_2))\delta^2 + o(\delta^3).$$

For job j in time interval $[t_1, t_1+\delta]$, similarly, we have

$$Q_j^{\text{new}}(t_1, t_1+\delta) = Q_j^*(t_1, t_1+\delta) - s'(\theta_j^*(t_1))s'(\theta_i^*(t_2))\delta^2 + o(\delta^3).$$

In the interval $[t_2, t_2+\delta]$, we decrease the service rate for job i by $\Delta_2 = s'(\theta_i^*(t_1))\delta$ and increase the service rate for job j by the same amount:

$$\theta_i^{\text{new}}(t) = \theta_i^*(t) - \Delta_2,\ \theta_j^{\text{new}}(t) = \theta_j^*(t) + \Delta_2, \quad \text{for } t \in [t_2, t_2+\delta].$$

Then for job i in time interval $[t_2, t_2+\delta]$, we have

$$Q_i^{\text{new}}(t_2, t_2+\delta) = Q_i^*(t_2, t_2+\delta) - s'(\theta_i^*(t_2),)s'(\theta_i^*(t_1))\delta^2 + o(\delta^3)$$

For job j in time interval $[t_2, t_2+\delta]$, we have

$$Q_j^{\text{new}}(t_2, t_2+\delta) = Q_j^*(t_2, t_2+\delta) + s'(\theta_j^*(t_2))s'(\theta_i^*(t_1))\delta^2 + o(\delta^3).$$

Combining both intervals, we have:

$$Q_i^{\text{new}}(t_1, t_1+\delta) + Q_i^{\text{new}}(t_2, t_2+\delta) - Q_i^*(t_1, t_1+\delta) - Q_i^*(t_2, t_2+\delta) = o(\delta^3),$$
$$Q_j^{\text{new}}(t_1, t_1+\delta) + Q_j^{\text{new}}(t_2, t_2+\delta) - Q_j^*(t_1, t_1+\delta) - Q_j^*(t_2, t_2+\delta) = h_{i,j}\delta^2 + o(\delta^3).$$

Therefore, the total service to job i changes by $o(\delta^3)$, while the service to job j increases by $h_{i,j}\delta^2 + o(\delta^3)$. Recall that $h_{i,j} > 0$. This implies that the completion time T_i changes by $o(\delta^3)$, and T_j decreases by a factor on the order of δ^2. That is, for a small δ, the new scheduling policy $\theta_i^{\text{new}}(t), \theta_j^{\text{new}}(t)$ yields a strictly smaller objective value than the optimal $(\theta_i^*(t), \theta_j^*(t))$, contradicting to the assumption of optimality. This completes the proof. □

The CDR Rule reveals a fundamental structural property of the optimal scheduling solution: the marginal benefit of allocating additional resource, measured by the derivative of the speedup function, must maintain a constant ratio across any pair of active jobs over time. This condition ensures that no further local reallocation of resources between jobs can improve the system performance. The

contradiction-based proof illustrates this by constructing a small resource reallocation between two jobs at two distinct time intervals. If the derivative ratio were not constant, such a reallocation would strictly reduce the objective function, violating optimality.

For the problem studied in [17], where the speedup function is $s(\theta) = \theta^p$ $(0 < p < 1)$, the authors established a "scale-free" property (Theorem 3 in [17]). That is, the ratio of service rates, $\frac{\theta_i^*(t)}{\theta_j^*(t)}$ between any two active jobs i and j remains constant over time t. This property can be viewed as a special case of the CDR Rule: for speedup functions $s(\theta) = \theta^p$, a constant ratio of $\frac{\theta_i^*(t)}{\theta_j^*(t)}$ is equivalent to a constant ratio of $\frac{s'(\theta_i^*(t))}{s'(\theta_j^*(t))}$.

Note that in Theorem 1, the value of $c_{i,j}$ is determined by any time instance t such that $\theta_i^*(t) > 0$ and $\theta_j^*(t) > 0$. If there does not exist any t such that $\theta_i^*(t) > 0$ and $\theta_j^*(t) > 0$, the value of $c_{i,j}$ can be set arbitrarily.

We now extend the analysis of Theorem 1 to the case where one job is allocated zero resource. Let T_i^* denote the completion time of job i under the optimal schedule $\theta_i^*(t)$. For any pair of jobs i and j, the condition $t < \min\{T_i^*, T_j^*\}$ ensures that both jobs remain incomplete under the optimal scheduler at time t. The following theorem establishes the structural property for time instances where $\theta_i^*(t) > 0$ and $\theta_j^*(t) = 0$.

Theorem 2. *For any pair of jobs i, j and two time instances t_1, t_2 with $\max\{t_1, t_2\} < \min\{T_1^*, T_2^*\}$, if $\theta_i^*(t_1) > 0$, $\theta_j^*(t_1) > 0$, $\theta_i^*(t_2) > 0$, $\theta_j^*(t_2) = 0$, then we have*

$$c_{i,j} = \frac{s'(\theta_i^*(t_1))}{s'(\theta_j^*(t_1))} \le \frac{s'(\theta_i^*(t_2))}{s'(0)}, \tag{7}$$

where $c_{i,j}$ is the constant given by Theorem 1.

Proof. We prove Theorem 2 by contradiction. Assume the theorem is false. Then we have

$$\frac{s'(\theta_i^*(t_1))}{s'(\theta_j^*(t_1))} > \frac{s'(\theta_i^*(t_2))}{s'(\theta_j^*(t_2))}.$$

Following the same approach in the proof of Theorem 1, in the interval $[t_1, t_1 + \delta]$, we increase the service rate for job i by a small amount $\Delta_1 = s'(\theta_i^*(t_2))\delta$ and decrease the service rate for job j by the same amount; in the interval $[t_2, t_2 + \delta]$, we decrease the service rate for job i by $\Delta_2 = s'(\theta_i^*(t_1))\delta$ and increase the service rate for job j by the same amount. We can construct a new scheduling solution $\theta_i^{\text{new}}(t)$ and $\theta_j^{\text{new}}(t)$ that decreases the objective value, which leads to a contradiction. □

Theorem 2 states that for any time t with $t < \min\{T_i^*, T_j^*\}$, if $\theta_i^*(t) > 0$ and $\theta_j^*(t) = 0$, then $\frac{s'(\theta_i^*(t))}{s'(0)} \ge c_{i,j}$. This result can be interpreted as follows. The CDR Rule (Theorem 1) requires the derivative ratio $\frac{s'(\theta_i^*(t))}{s'(\theta_j^*(t))}$ to equal $c_{i,j}$ whenever both $\theta_i^*(t)$ and $\theta_j^*(t)$ are positive. Since $s'(\theta) > 0$ and $s'(\theta)$ is decreasing with θ, consider a time t such that $s'(\theta_i^*(t)) > c_{i,j}s'(0)$. To satisfy the CDR Rule, we would need $s'(\theta_j^*(t)) = \frac{s'(\theta_i^*(t))}{c_{i,j}} > s'(0)$. However, because $s'(\theta)$ decreases with θ, this would imply $\theta_j^*(t) < 0$, which is impossible. Thus, the non-negativity constraint enforces $\theta_j^*(t) = 0$, which leads to the condition in Theorem 2.

Note that in [17], where $s(\theta) = \theta^p$ with $0 < p < 1$, we always have $\theta_i^*(t) > 0$ for any $t < T_i^*$. This occurs because $s'(0) = \infty$, implying that the benefit of allocating a small amount of resource to any unfinished job with zero scheduling rate is infinite. As a result, under the optimal scheduling, every job must receive a positive allocation until its completion. In contrast, in the general case considered in this paper, where $s'(0) < \infty$, it is possible (and in fact quite common) for some unfinished jobs to have zero scheduling rate. This behavior is precisely the case characterized in Theorem 2. We will revisit this phenomenon in the following sections.

Theorems 1 and 2 leads to the following corollary.

Corollary 2.1. *There exists M constants $c_1, c_2, \cdots, c_M$ such that*

$$\frac{s'(\theta_i^*(t))}{s'(\theta_j^*(t))} = \frac{c_i}{c_j}, \tag{8}$$

for all t such that $\theta_i^*(t) > 0$ and $\theta_j^*(t) > 0$.

Proof. We prove Corollary 2.1 by induction. We assume the optimal job completion time has an order $T_1^* > T_2^* > \cdots > T_N^*$ [2]. By definition in Theorem 1, it is easy to prove that for any i and j, we have $c_{i,j} = \frac{1}{c_{j,i}}$. So by proving $c_{i,j} = \frac{c_i}{c_j}$ we will automatically have $c_{j,i} = \frac{c_j}{c_i}$.

For jobs 1 and 2, by Theorem 1 there exists $c_{2,1}$. We can arbitrarily set c_1 and set $c_2 = c_{2,1}c_1$.

For jobs $1, 2, \cdots, k$ $(k \ge 2)$, suppose we can find $c_1, c_2, \cdots, c_k$ such that $c_{i,j} = \frac{c_i}{c_j}$ for $i < j \le k$. Then consider job $k+1$. If there does not exist a job $i \le k$ and a time t making $\theta_i^*(t) > 0$ and $\theta_{k+1}^*(t) > 0$, we can arbitrarily set c_{k+1}. If there exists a job $i \le k$ and a time t making $\theta_i^*(t) > 0$ and $\theta_{k+1}^*(t) > 0$, we set $c_{k+1} = \frac{c_i}{c_{i,k+1}}$. All we need to prove is that for all $j \ne i$, we have $c_{k+1} = c_{k+1,j}c_j$. We discuss the two cases. At time t, if $\theta_j^*(t) > 0$, then since $\frac{s'(\theta_i^*(t))}{s'(\theta_j^*(t))} = \frac{c_i}{c_j}$ and $\frac{s'(\theta_{k+1}^*(t))}{s'(\theta_i^*(t))} = \frac{c_{k+1}}{c_i}$, we have $= c_{k+1,j} = \frac{s'(\theta_{k+1}^*(t))}{s'(\theta_j^*(t))} = \frac{c_{k+1}}{c_j}$, indicating that $c_{k+1} = c_{k+1,j}c_j$. If $\theta_j^*(t) = 0$, then by Theorem 2, we have $\frac{c_i}{c_j} \le \frac{s'(\theta_i^*(t))}{s'(0)} < 1$. Then for any $\tau < T_{k+1}^*$ (by the order of job completion, at time τ jobs $i, j, k+1$ are not completed) such that $\theta_j^*(\tau) > 0$ and $\theta_{k+1}^*(\tau) > 0$, since $\frac{c_i}{c_j} < 1$, we have $\theta_i^*(\tau) > 0$. So we have $\frac{s'(\theta_i^*(\tau))}{s'(\theta_j^*(\tau))} = \frac{c_i}{c_j}$ and $\frac{s'(\theta_{k+1}^*(\tau))}{s'(\theta_i^*(\tau))} = \frac{c_{k+1}}{c_i}$, indicating that $c_{k+1} = c_{k+1,j}c_j$. This completes our proof. □

Compared with Theorem 1, Corollary 2.1 further reduces the space of c's. We only need M constants c_i's instead of $\frac{M(M-1)}{2}$ constants $c_{i,j}$'s.

In summary, in this section we identify the key "consistent derivative ratio" property associated with the optimal solution to OPT. These results significantly reduce the search space: to find the optimal solution, it is sufficient to consider only the schedulers that satisfy these properties.

The next question is how to construct schedulers that satisfy this property. We address this by studying the problem of determining the allocations under fixed derivative ratios in the next section. After that, we will present our complete solution to OPT.

[2]This assumption is always true for OPT (see Proposition 5) and we use it to simplify notations. Without this assumption, the proof of Corollary 2.1 still holds after we re-order the jobs by their completion times.

4 GENERAL WATER-FILLING ALGORITHM

In this section, we study the problem of determining the resource allocation at a given time instance under fixed derivative ratios. The solution to this problem will serve as a fundamental component in our overall algorithm for solving OPT.

4.1 Constrained Allocation Problem Statement

We refer to this problem as the *Constrained Allocation Problem (CAP)*. Given a speedup function $s(\theta)$, a total allocation budget $0 < b \le B$, an integer $k \ge 2$, and k constants $c_1 \ge c_2 \ge \cdots \ge c_k > 0$, CAP is defined as follows:

$$\begin{aligned}
\text{CAP:} \quad & \text{Find } \theta_1, \theta_2, \cdots, \theta_k && (9)\\
\text{subject to} \quad & \theta_1 + \theta_2 + \cdots + \theta_k = b, && (9a)\\
& \theta_1 \le \theta_2 \le \cdots \le \theta_k, && (9b)\\
& \frac{s'(\theta_j)}{s'(\theta_i)} = \frac{c_j}{c_i}, \quad \text{if } i < j \text{ and } \theta_j \ge \theta_i > 0, && (9c)\\
& \frac{s'(\theta_j)}{s'(0)} \ge \frac{c_j}{c_i}, \quad \text{if } i < j \text{ and } \theta_j > \theta_i = 0. && (9d)
\end{aligned}$$

Here, constraint (9a) enforces that the sum of allocations $\theta_1, \theta_2, \ldots, \theta_k$ equals b; constraint (9b) requires the allocations to be ordered; and constraints (9c) and (9d) restrict the ratios of the derivatives by c_i's.

4.2 Algorithm to Solve CAP

In this subsection, we present our algorithm to solve CAP.

Let $s'^{(-1)}(y)$ denote the inverse function of $s'(\theta)$. The domain of $s'^{(-1)}(y)$ is $y \in [s'(b), s'(0)]$ (if $s'(0) = \infty$, then $y \in [s'(b), \infty)$). Since $s'(\theta)$ is continuous and strictly decreasing w.r.t. θ, $s'^{(-1)}(y)$ is also continuous and strictly decreasing w.r.t. y. Besides, we have $s'^{(-1)}(y) \ge 0$.

We define an *auxiliary function*, denoted by $g(h)$, to help solve CAP. Let $[h_{\min}, h_{\max}]$ denote the domain of $g(h)$ The function $g(h)$ and $[h_{\min}, h_{\max}]$ must satisfy the following conditions:

$$g(h) \text{ is continuous and strictly decreasing w.r.t. } h. \tag{10a}$$

$$g(h_{\max}) \le \frac{s'(b)}{c_1}. \tag{10b}$$

$$\text{If } s'(0) = \infty, \text{ then } g(h_{\min}) = \infty. \text{ If } s'(0) < \infty, \text{ then } g(h_{\min}) \ge \frac{s'(0)}{c_k}. \tag{10c}$$

For each CAP problem, we need to select an auxiliary function $g(h)$ and an interval $[h_{\min}, h_{\max}]$ such that conditions (10a), (10b), and (10c) are satisfied. Examples of auxiliary functions that work for all CAP problems include: $g(h) = -h$ with $h_{\min} = -\infty$, $h_{\max} = \infty$; and $g(h) = \frac{1}{h}$ with $h_{\min} = 0$, $h_{\max} = \infty$. In fact, there exists a broad class of feasible auxiliary functions, and different choices can be made depending on the specific CAP instance. We will discuss the selection of $g(h)$ and $[h_{\min}, h_{\max}]$ in the following subsections.

Given $g(h)$, for each $i = 1, 2, \cdots, k$, we define the function $\theta_i(h)$ as:

$$\theta_i(h) = \begin{cases} 0 & \text{if } c_i g(h) \ge s'(0), \\ s'^{(-1)}\big(c_i g(h)\big) & \text{if } s'(b) < c_i g(h) < s'(0), \\ b & \text{if } c_i g(h) \le s'(b). \end{cases} \tag{11}$$

It can be shown that for each $i = 1, 2, \ldots, k$, $\theta_i(h)$ is continuous and increasing in $h \in [h_{\min}, h_{\max}]$. We refer to $\theta_i(h)$ as the *water-filling function* for job i, since it can be interpreted as the "water volume in bottle i when the water level is h." This analogy will be further discussed in Section 4.4.

Let $\beta(h)$ denote the sum of $\theta_i(h)$'s across all jobs:

$$\beta(h) = \theta_1(h) + \theta_2(h) + \cdots + \theta_k(h). \tag{12}$$

In the water-filling analogy of Section 4.4, $\beta(h)$ represents the "total water volume across all bottles when the water level is h."

We consider the following problem, which is referred as *Water-Filling Problem (WFP)*.

$$\begin{aligned}
\text{WFP:} \quad & \text{Find } h && (13)\\
\text{subject to} \quad & \beta(h) = b. && (13a)
\end{aligned}$$

The following proposition states the existence of a solution to WFP.

Proposition 1. *WFP has a unique solution.*

Proof. It can be shown that $\beta(h)$ is continuous and increasing. Since $\beta(h_{\min}) = 0 < b$ and $\beta(h_{\max}) = kb > b$, there exists a unique h making $\beta(h) = b$. This completes our proof. □

The following proposition states the a solution of WFP is also a solution to CAP.

Proposition 2. *If h is a solution to WFP, then $\theta_i = \theta_i(h)$ for $i = 1, 2, \cdots, k$ is a solution to CAP.*

Proof. We need to verify this solution $\theta_i = \theta_i(h)$ satisfies all the constraints in CAP.

By (13a), we have $\theta_1 + \theta_2 + \cdots + \theta_k = b$. So the solution satisfies (9a).

It can be shown that $g(h) > 0$ (otherwise $\theta_i(h) = b$ for all i's and (13a) cannot be satisfied).

For any $i < j$, we have $c_i g(h) \ge c_j g(h)$. By definition in (11), we have $\theta_i(h) \le \theta_j(h)$. So the solution satisfies (9b).

Consider any $i < j$ and $\theta_i(h) > 0$. We have $0 < \theta_i(h) < \theta_j(h) < b$. And we have $\theta_i(h) = s'^{(-1)}\big(c_i g(h)\big)$ and $\theta_j(h) = s'^{(-1)}\big(c_j g(h)\big)$. And we have $\frac{s'(\theta_j)}{s'(\theta_i)} = \frac{c_j g(h)}{c_i g(h)} = \frac{c_j}{c_i}$. So the solution satisfies (9c).

Consider any $i < j$, $\theta_i(h) = 0$, and $\theta_j(h) > 0$. We have $c_i g(h) \ge s'(0)$ and $\theta_j(h) = s'^{(-1)}\big(c_j g(h)\big)$. And we have $\frac{s'(\theta_j)}{s'(\theta_i)} = \frac{c_j g(h)}{s'(0)} \ge \frac{c_j g(h)}{c_i g(h)} = \frac{c_j}{c_i}$. So the solution satisfies (9d). This completes our proof. □

The following proposition states the number of solutions to CAP.

Proposition 3. *CAP has at most one solution.*

Proof. We prove Proposition 3 by contradiction. Suppose CAP has two different solutions $\theta_1, \theta_2, \cdots, \theta_k$ and $\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_k$. Since $\theta_1 + \theta_2 + \cdots + \theta_k = \hat{\theta}_1 + \hat{\theta}_2 + \cdots + \hat{\theta}_k = b$, without loss of generality, we can find $i < j$ making $\hat{\theta}_i < \theta_i \le \theta_j < \hat{\theta}_j$. Based on constraints (9c) and (9d), we have

$$\frac{s'(\hat{\theta}_j)}{s'(\hat{\theta}_i)} \ge \frac{c_j}{c_i} = \frac{s'(\theta_j)}{s'(\theta_i)}. \tag{14}$$

Algorithm 1 General Water-Filling (GWF) Algorithm to Solve CAP

Input: $s(\theta)$, b, k, and $c_1, c_2, \cdots, c_k$.
Output: The solution to CAP: $\theta_1^*, \theta_2^*, \cdots, \theta_k^*$.
1: Select an auxiliary function $g(h)$ that satisfies (10a), (10b), and (10c).
2: Solve WFP and get its solution h^*.
3: Compute $\theta_i^* = \theta_i(h^*)$ based on (11) for $i = 1, 2, \cdots, k$.

On the other hand, since $s'(\theta)$ is strictly decreasing and positive, we have $0 < s'(\hat{\theta}_j) < s'(\theta_j)$ and $0 < s'(\theta_i) < s'(\hat{\theta}_i)$. So we have

$$\frac{s'(\hat{\theta}_j)}{s'(\hat{\theta}_i)} < \frac{s'(\theta_j)}{s'(\theta_i)}. \tag{15}$$

We can see (14) and (15) leads to a contradiction. This completes our proof. □

Combining Propositions 1, 2, and 3, we have the following theorem.

Theorem 3. *CAP has a unique solution, which is the solution of WFP.*

Based on Theorem 3, we propose the *General Water Filling (GWF)* algorithm to solve CAP, as shown in Algorithm 1.

Note that in Step 1 of Algorithm 1, we can select $g(h)$ from a large class of feasible auxiliary functions. However, a good selection of $g(h)$ can make WFP much easier to solve, i.e., make Step 2 of Algorithm 1 easier. We will use an example to show this point in the next subsection.

4.3 An Example for Selecting $g(h)$

Consider a speedup function: $s(\theta) = a(\theta + z)^p - az^p$, where $a > 0$, $z \geq 0$, and $0 < p < 1$ are constants. Note that when $z = 0$, the speedup function $s(\theta) = a\theta^p$ is the one considered in [17]. We will use this speedup function as an example to show how to select $g(h)$ to make WFP easier to solve.

For this speedup function, we have $s'(\theta) = ap(\theta + z)^{p-1}$ and $s'^{(-1)}(y) = (\frac{y}{ap})^{\frac{1}{p-1}} - z$. Consider (11). When $s'(b) < c_i g(h) < s'(0)$, for each $i = 1, 2, \cdots, k$ we have

$$\theta_i(h) = s'^{(-1)}\big(c_i g(h)\big) = (\frac{c_i g(h)}{ap})^{\frac{1}{p-1}} - z. \tag{16}$$

If we select $g(h) = aph^{p-1}$ and $h_{\min} = 0$, $h_{\max} = \infty$, then (16) becomes:

$$\theta_i(h) = c_i^{\frac{1}{p-1}} h - z = c_i^{\frac{1}{p-1}} (h - zc_i^{\frac{1}{1-p}}),$$

and for each $i = 1, 2, \cdots, k$ we have

$$\theta_i(h) = \begin{cases} 0 & \text{if } h \leq zc_i^{\frac{1}{1-p}}, \\ c_i^{\frac{1}{p-1}} (h - zc_i^{\frac{1}{1-p}}) & \text{if } zc_i^{\frac{1}{1-p}} < h < (z+b)c_i^{\frac{1}{1-p}}, \\ b & \text{if } h \geq (z+b)c_i^{\frac{1}{1-p}}. \end{cases} \tag{17}$$

We can see the selection of $g(h) = aph^{p-1}$ will give us a piecewise linear $\theta_i(h)$ for all i's, making WFP easy to solve. This means $g(h) = aph^{p-1}$ is a good selection.

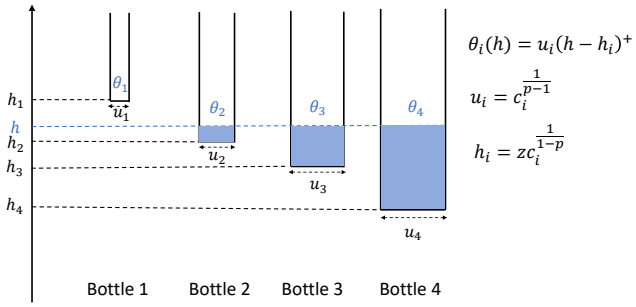

Figure 2: An illustration for GWF Algorithm when $s(\theta) = a(\theta + z)^p - az^p$ and $k = 4$.

On the other hand, if we select $g(h) = ap\log(1+\frac{1}{h})$ and $h_{\min} = 0$, $h_{\max} = \infty$, then (16) becomes: and for each $i = 1, 2, \cdots, k$ we have

$$\theta_i(h) = \begin{cases} 0 & \text{if } c_i g(h) \geq s'(0), \\ \big(c_i \log(1 + \frac{1}{h})\big)^{\frac{1}{p-1}} - z & \text{if } s'(b) < c_i g(h) < s'(0), \\ b & \text{if } c_i g(h) \leq s'(b). \end{cases} \tag{18}$$

This is non-linear and makes WFP not easy to solve. This means $g(h) = ap\log(1 + \frac{1}{h})$ is not a good selection.

4.4 Water-Filling Illustration

Water-filling algorithm is a classic algorithm for solving the power allocation problem in multi-channel communication systems. As illustrated in Fig. 5.11 in [18], the algorithm can be visualized as pouring "water" (i.e., power) into parallel subcarriers such that the water level across all subcarriers containing water is equalized. Each subcarrier has the same "width" but a different "bottom height," determined by its channel quality. The water-filling algorithm is proven to be optimal for power allocation in multi-channel communication. Further details can be found in Chapter 5.3.3 of [18].

We observe that in our GWF algorithm for solving SAP, the allocation result can be naturally interpreted through the water-filling analogy. Fig. 2 illustrates the GWF algorithm when $s(\theta) = a(\theta+z)^p - az^p$ and $k = 4$. In this case, we construct $k = 4$ "rectangle" bottles, each with its own "width" and "bottom height."

Let u_i and h_i denote the width and bottom height of bottle i, respectively, and let $\theta_i(h)$ denote the volume of water in bottle i when the water level is h. We set $u_i = c_i^{\frac{1}{p-1}}$ and $h_i = zc_i^{\frac{1}{1-p}}$.

Define $(x)^+ = \max(0, x)$. From (17), we have $\theta_i(h) = u_i(h - h_i)^+$ whenever $\theta_i(h) < b$. Thus, solving WFP reduces to finding the water level h in Fig. 2 such that the total water volume across all bottles equals b.

This is directly analogous to the classical water-filling algorithm in multi-channel communication with one important generalization: in GWF (Fig. 2), the bottle widths differ across jobs, whereas in the classical water-filling algorithm, all channel widths are identical. Note that when $z = 0$, all bottles have the same bottom heights, which indicates for any $b > 0$, $\theta_i^* > 0$ for all i's.

A more general illustration for GWF is shown in Fig. 3, where the bottles are not rectangle-shape. Here we define $\theta_i(h) = (\rho_i(h))^+$,

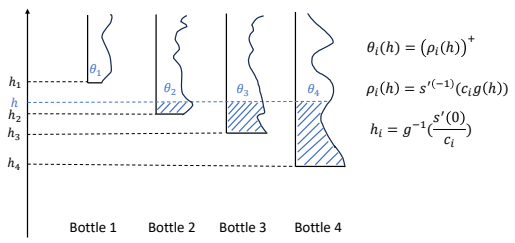

Figure 3: An illustration for GWF Algorithm for non-rectangle bottles.

Table 1: Some examples of regular speedup functions

Regular Speedup Function
$s(\theta) = a(\theta+z)^p - az^p, \theta \in [0, B]$, where $a > 0$, $z \geq 0$, and $0 < p < 1$
$s(\theta) = a\ln(p\theta+1), \theta \in [0, B]$, where $a > 0$ and $p > 0$
$s(\theta) = az^p - a(\theta+z)^p, \theta \in [0, B]$, where $a > 0$, $z \geq 0$, and $p < 0$
$s(\theta) = az^p - a(z-\theta)^p, \theta \in [0, B]$, where $a > 0$, $z \geq B$, and $p > 1$

where $\rho_i(h) = s'^{(-1)}(c_i g(h))$. Let $g^{-1}(\cdot)$ denote the inverse function of $g(h)$. And we define $h_i = g^{-1}(\frac{s'(0)}{c_i})$. We can see $\theta_i(h)$ in (11) can be illustrated as filling water in bottles, as shown in Fig. 3. When $s'(0) = \infty$, we have $h_1 = h_2 = \cdots = h_k$, indicating that all bottles have the same bottom height.

We note that variants of the water-filling algorithm have been widely developed in the literature to address static optimization problems. For example, [19] extends water-filling to handle weighted-sum-rate, weighted-sum-power, and peak-power optimization in multi-channel communication systems. Similarly, [20] proposes a generalized water-filling approach for solving concave maximization problems under power constraints. In contrast to these works, which focus on static settings, our proposed GWF is tailored to a dynamic resource allocation problem. Moreover, our algorithmic framework—particularly the introduction of the auxiliary function $g(h)$ and the notion of "regular" speedup functions—is fundamentally new and proves to be highly effective.

4.5 Solving WFP

In this subsection, we discuss how to solve WFP, i.e., how to run Step 2 in Algorithm 1.

4.5.1 Closed-Form Solution. In the ideal case, we can get a closed-form solution to WFP. In this paper we identify a class of speedup functions $s(\theta)$, which we call *regular functions*. Under regular $s(\theta)$, we can get a closed-form solution to WFP, and thus get a closed-form solution to CAP by Algorithm 1.

Definition 4. A speedup function $s(\theta)$ is regular if there exists $\alpha \neq 0, \gamma \neq 0$ and $z \in \mathcal{R}$ such that $s'(\theta) = \alpha(\theta+z)^\gamma$ for all $\theta \in [0, B]$.

Some examples of regular speedup functions are given in Table 1. Readers can verify each row in Table 1 satisfies Definition 4. We can see the speedup function $s(\theta) = a\theta^p$ $(0 < p < 1)$ considered in [17] is regular.

For regular $s(\theta)$ such that $s'(\theta) = \alpha(\theta+z)^\gamma$, we have a closed-form expression for $s'^{(-1)}(y)$:

$$s'^{(-1)}(y) = (\frac{y}{\alpha})^{\frac{1}{\gamma}} - z.$$

We can select $g(h) = (\alpha h)^\gamma$. Based on 11, we have $\theta_i(h) = u_i(h-h_i)^+$ whenever $\theta_i(h) < b$, where $u_i = c_i^{\frac{1}{\gamma}}$ and $h_i = zc_i^{-\frac{1}{\gamma}}$. In the water-filling illustration, the bottles are rectangles. And the problem WFP is:

WFP under regular $s(\theta)$: Find h

$$\text{subject to} \quad \beta(h) = \sum_{i=1}^{k} u_i(h-h_i)^+ = b.$$

Notice that $\beta(h)$ is a piecewise linear function. WFP under regular $s(\theta)$ can be solved easily. We omit the details here.

For general non-regular $s(\theta)$, whether we can select a good $g(h)$ to get an closed-form expression of the solution to WFP is unclear to us. We believe for functions like $s(\theta) = \theta^{\frac{1}{2}} + \ln(1+\theta)$, it is hard to write down a closed-form expression for $s'^{(-1)}(y)$, and thus it is very difficult to get an closed-form solution to WFP.

4.5.2 Numerical Methods. For non-regular speedup functions that we don't have closed-form solutions, we can always use numerical methods to get a numerical solution. For example, we can arbitrarily select a feasible $g(h)$ (e.g., select $g(h) = -h$ with $h_{\min} = -\infty$, $h_{\max} = \infty$), and numerically compute $\beta(h)$ by (12). When $\beta(h) < b$, we increase h; when $\beta(h) > b$, we decrease h, until the gap between $\beta(h)$ and b is small enough. This can be done by methods like bisection [22] to solve it.

5 SMARTFILL ALGORITHM

Based on the CDR Rule and the GWF algorithm, in this section we present the *SmartFill* Algorithm, which is shown to be able to solve OPT optimally.

5.1 Preliminary Results

Before presenting SmartFill, we introduce some preliminary results.

Let $m^*(t)$ denote the number of jobs in the system at time t under the optimal scheduling policy. We begin with the following proposition:

Proposition 4. *For any two time instances t_1 and t_2, if $m^*(t_1) = m^*(t_2)$, then $\theta_i^*(t_1) = \theta_i^*(t_2)$ for all $i = 1, 2, \cdots, M$.*

Proposition 4 is adapted from Theorem 1 in [17]. The proof of Proposition 4 is identical to the proof of Theorem 1 in [17].

Regarding the job completion order, we have the following proposition:

Proposition 5. *The optimal policy completes jobs in Shortest-Job-First (SJF) order: $M, M-1, \cdots, 1$.*

The proof of Proposition 5 is identical to the proof of Theorem 2 in [17]. Let T_i^* denote the completion time for job i $(i = 1, 2, \cdots, M)$ under the optimal policy. By Proposition 5, we have

$$T_1^* > T_2^* > \cdots > T_M^*. \tag{20}$$

Based on Propositions 4 and 5, let θ_i^j denote the constant scheduling rate allocated to job i during the time interval $[T_{j+1}^*, T_j^*)$, where

we define $T^*_{M+1} = 0$ for convenience. We then define the scheduling matrix $\Theta = [\theta_i^j]$ as

$$\Theta = \begin{bmatrix} \theta_1^1 & \theta_1^2 & \cdots & \theta_1^M \\ \theta_2^1 & \theta_2^2 & \cdots & \theta_2^M \\ \vdots & \vdots & \ddots & \vdots \\ \theta_M^1 & \theta_M^2 & \cdots & \theta_M^M \end{bmatrix}. \tag{21}$$

Clearly, finding the optimal scheduling policy is equivalent to determining the optimal matrix Θ.

The resource constraint in (1) translates into the following condition on the scheduling matrix:

$$\sum_{i=1}^{M} \theta_i^j = B, \quad \forall j = 1, 2, \cdots, M. \tag{22}$$

In (22), we use equality rather than inequality because, under the optimal policy, the total available resource should be fully utilized at all times.

Since the optimal scheduler follows the SJF order, we have $\theta_i^j = 0$ for all $i > j$. In other words, the scheduling matrix Θ is upper triangular:

$$\Theta = \begin{bmatrix} \theta_1^1 & \theta_1^2 & \cdots & \theta_1^M \\ 0 & \theta_2^2 & \cdots & \theta_2^M \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \theta_M^M \end{bmatrix}. \tag{23}$$

5.2 Main Algorithm

The idea of SmartFill is to iteratively compute the optimal scheduling results from the last completed job (job 1) to the first completed job (job M). After each iteration k, we record the c_i's ($1 \le i \le k$) for the CDR rule, and use them to compute the optimal scheduling results for iteration $k+1$.

For each $b \le B$ and $1 \le i \le k$, we define the CAP function

$$\mathrm{CAP}_i(b, c_1, c_2, \cdots, c_k) \tag{24}$$

as the optimal solution θ_i for the CAP problem. The value of CAP function can be computed by the GWF algorithm (Algorithm 1).

The following propositions states that the optimal objective is a linear combination of x_i's.

Proposition 6. *Under fixed w_i's, $s(\theta)$, and B, for different x_i's such that $x_1 \ge x_2 \ge \cdots \ge x_M$, the optimal objective value can be expressed as*

$$J^* = \sum_{i=1}^{M} a_i x_i, \tag{25}$$

where the coefficients satisfy $a_1 < a_2 < \cdots < a_M$, and a_i's are independent of the specific values of x_i.

A Proof Sketch We prove Proposition 6 by induction. On one hand, it is apparent that Proposition 6 holds for $M = 1$.

On the other hand, assume Proposition 6 holds for $M = k$ ($k \ge 1$). Then consider the case $M = k+1$. Consider the time instance T_M when job M completes. After T_M, the optimal weighted sum of completion times can be written as $J' = \sum_{i=1}^{k} \alpha_i x_i'$, where x_i' is the remaining size for job i at $t = T_M$. And the parameters $c_1, c_2, \cdots, c_k$ can be determined. Then for time interval $[0, T_M]$, when the service rate for job $M = k+1$ is $\theta_M^M = \mu$, the service rate for job i ($i \le k$) is $\theta_i^M = \mathrm{CAP}_i(B - \mu, c_1, c_2, \cdots, c_k)$. The optimal solution of μ is given by the following problem:

$$\arg\min_{\mu} \Big(\sum_{i=1}^{k+1} w_i\Big) \cdot \frac{x_{k+1}}{s(\mu)} - \Big(\sum_{i=1}^{k} a_i s(\mathrm{CAP}_i(B-\mu, c_1, c_2, \cdots, c_k))\Big) \cdot \frac{x_{k+1}}{s(\mu)}.$$

It is easy to see that the value of μ is independent from the value of x_{k+1}. So J^* is also linear with x_{k+1}. This completes the proof. ∎

A complete proof of Proposition 6 can be found in Appendix A.

Now we present the SmartFill Algorithm. For iteration $k = 1$, we set $\theta_1^1 = B$, $c_1 = 1$, and $a_1 = \frac{w_1}{s(B)}$. After iteration k, we will have $c_1, c_2, \cdots, c_k$ and $a_1, a_2, \cdots, a_k$. For iteration $k+1$, we compute θ_{k+1}^{k+1} by

$$\theta_{k+1}^{k+1} = \arg\min_{\mu} \ \frac{\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\mathrm{CAP}_i(B-\mu, c_1, c_2, \cdots, c_k))}{s(\mu)}. \tag{26}$$

Then we compute θ_i^{k+1} for $i = 1, 2, \cdots, k$ by

$$\theta_{k+1}^i = \mathrm{CAP}_i(B - \theta_{k+1}^{k+1}, c_1, c_2, \cdots, c_k). \tag{27}$$

Then we compute c_{k+1} and a_{k+1} by:

$$c_{k+1} = \frac{s'(\theta_{k+1}^{k+1})}{s'(\theta_k^{k+1})} c_k, \tag{28}$$

and

$$a_{k+1} = \frac{\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\theta_i^{k+1})}{s(\theta_{k+1}^{k+1})}. \tag{29}$$

Algorithm 2 SmartFill Algorithm to Solve OPT

Input: $s(\theta)$, B, M, x_i's, and w_i's.
Output: The optimal solution $\Theta = [\theta_i^j]$ to OPT.

1: Set $\theta_i^j = 0$ for all $i > j$.
2: Set $\theta_1^1 = B$, $c_1 = 1$, and $a_1 = \frac{w_1}{s(B)}$.
3: **for** $k = 1, 2, \cdots, M-1$ **do**
4: Compute θ_{k+1}^{k+1} by (26).
5: Compute θ_i^{k+1} for $i = 1, 2, \cdots, k$ by (27).
6: Compute c_{k+1} and a_{k+1} by (28) and (29).
7: **end for**

The complete procedure of SmartFill is summarized in Algorithm 2. Note that in Step 4, we need to solve the maximization problem defined in (26). When $s(x)$ is a regular speedup function, the GWF algorithm ensures that $\mathrm{CAP}_i(B-\mu, c_1, c_2, \ldots, c_k)$ is linear in μ for each i, allowing for a closed-form solution. When $s(x)$ is non-regular, this step can be performed numerically based on the GWF procedure. Due to space constraints, the details of this step are omitted.

The following theorem establishes the optimality of SmartFill.

Theorem 5. *SmartFill (Algorithm 2) provides an optimal solution to OPT.*

A complete proof of Theorem 5 can be found in Appendix B.

Recall that heSRPT is optimal for a special family of speedup functions $s(\theta) = \theta^p$ with $0 < p < 1$ [17]. For this family of functions, SmartFill produces the same allocation as heSRPT, which will be validated in the numerical results presented in the next section.

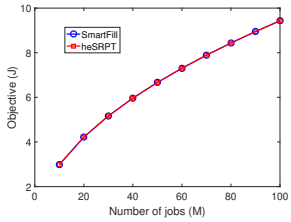

Figure 4: Mean slowdown when $s(\theta) = \theta^{0.5}$.

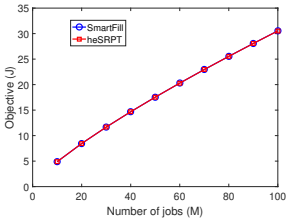

Figure 5: Mean slowdown when $s(\theta) = 10\theta^{0.8}$.

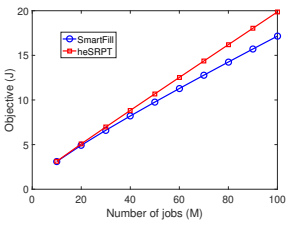

Figure 6: Mean slowdown when $s(\theta) = \log(1+\theta)$.

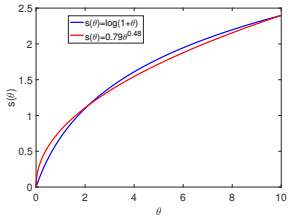

Figure 7: Approximation for $s(\theta) = \log(1+\theta)$ used by heSRPT.

Unlike heSRPT, which allocates resources to all active jobs, SmartFill intelligently determines which jobs should receive resources and how much they should be allocated. This structural difference arises from the behavior of $s'(0)$. When $s'(0) = \infty$, as in the case of $s(\theta) = \theta^p$, allocating even a small amount of resource to any unfinished job yields infinite benefit, implying that all jobs must receive positive allocation until completion. In contrast, in the general case where $s'(0) < \infty$, it is possible for some unfinished jobs to have zero scheduling rate. SmartFill identifies these jobs by solving the optimization problem in (26).

6 NUMERICAL RESULTS

In this section we provide some numerical results for SmartFill. We use heSRPT [17] as the performance benchmark.

6.1 Special Case: $s(\theta) = a\theta^p$

We first consider the special case $s(\theta) = a\theta^p$, where heSRPT is optimal [17]. We consider $M = 10, 20, \cdots, 100$ jobs. We consider $B = 10$. For each M, we assume $x_1 = M, x_2 = M-1, \cdots, x_M = 1$. We consider the mean slowdown at the objective J, i.e., $w_i = \frac{1}{x_i}$. Figs. 4 and 5 shows the mean slowdown under different M's when $s(\theta) = \theta^{0.5}$ and $s(\theta) = 10\theta^{0.8}$, respectively. In all cases, SmartFill achieves the same mean slowdown as heSRPT, thereby validating that SmartFill is optimal in this setting.

6.2 General Concave $s(\theta)$

We next evaluate the performance of SmartFill under general concave speedup functions. In [17], the authors suggested that when $s(\theta) \neq a\theta^p$, the heSRPT policy can still be applied by approximating $s(\theta)$ with a function of the form $a\theta^p$. We adopt this approximation-based heSRPT as our performance benchmark.

As before, we consider systems with $M = 10, 20, \ldots, 100$ jobs and a total resource budget $B = 10$. For each M, the job sizes are set as $x_1 = M, x_2 = M-1, \ldots, x_M = 1$. The performance metric is again the mean slowdown, defined by the objective J with $w_i = \frac{1}{x_i}$.

Fig. 6 shows the mean slowdowns of SmartFill and heSRPT when $s(\theta) = \log(1+\theta)$. Here, heSRPT uses the approximation $s(\theta) = 0.79\theta^{0.48}$, shown in Fig. 7. SmartFill consistently outperforms heSRPT, achieving a 13.6% lower slowdown at $M = 100$.

Fig. 8 shows results for $s(\theta) = \sqrt{4+\theta} - 2$, where heSRPT employs the approximation $s(\theta) = 0.26\theta^{0.82}$, as shown in Fig. 9. Again, SmartFill outperforms heSRPT. Because this approximation is tighter than

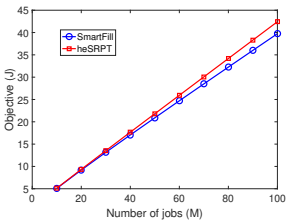

Figure 8: Mean slowdown when $s(\theta) = \sqrt{4+\theta} - 2$.

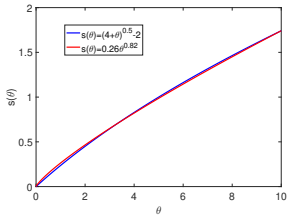

Figure 9: Approximation for $s(\theta) = \sqrt{4+\theta} - 2$ used by heSRPT.

in the previous case, the performance gap is smaller: at $M = 100$, SmartFill's slowdown is 6.3% lower than that of heSRPT.

Similar trends are observed across a variety of concave speedup functions. SmartFill consistently outperforms heSRPT whenever $s(\theta) \neq \theta^p$. The performance gap between SmartFill and heSRPT increases when the speedup function is further from the form θ^p.

7 DISCUSSION: TOWARDS MORE GENERAL PROBLEMS

This paper focuses on the case where all jobs share the same speedup function $s(\theta)$, the system resource budget B is constant over time, and the objective J is a weighted sum of job completion times. We presented the CDR Rule and the GWF method, and the SmartFill Algorithm to solve this problem. Beyond this specific setting, however, we find that the CDR Rule is applicable to much more general scenarios, including heterogeneous speedup functions across jobs, time-varying resource budgets, and more general system objectives J.

7.1 General Problem Definition

Consider a system with M parallelizable jobs, each available at time $t = 0$. Let x_i denote the size of job i for $i = 1, 2, \ldots, M$. Let $B(t)$ denote the total available resource at time t, and let $\theta_i(t)$ denote the allocation to job i at time t. The allocations must satisfy

$$\sum_{i=1}^{M} \theta_i(t) \leq B(t), \quad \forall t > 0. \tag{30}$$

Each job i is associated with a time-varying speedup function $s_i(\theta, t)$, which denotes its service rate at t when allocated resource θ. We assume that for each i and each fixed t, the function satisfies:

- $s_i(0,t) = 0$,
- $s_i(\theta,t)$ is strictly increasing w.r.t. θ,
- $s_i(\theta,t)$ is differentiable w.r.t. θ.
- $s_i(\theta,t)$ is strictly concave w.r.t. θ.

Let $s_i'(\theta,t) = \frac{\partial s_i(\theta,t)}{\partial \theta}$. We assume that $s_i'(\theta,t)$ is continuous in θ, and that $B(t)$, $s_i(\theta,t)$, $s_i'(\theta,t)$, and $\theta_i(t)$ are right-continuous in t.

The amount of service received by job i in the interval $[t_1,t_2]$ is

$$Q_i(t_1,t_2) = \int_{t_1}^{t_2} s_i(\theta_i(t),t)\,dt, \quad \forall t_1 < t_2. \tag{31}$$

Let T_i denote the completion time of job i. The system performance metric J is assumed to be a continuous and strictly increasing function of the completion times:

$$J = f(T_1, T_2, \ldots, T_M). \tag{32}$$

The goal is to find the optimal allocation $\theta_i(t)$ for each job i and time t that minimizes J, given $B(t)$, x_i, $s_i(\theta,t)$, and f.

7.2 CDR Rule for the General Problem

Let $\theta^*(t)$ denote an optimal scheduling policy for the general problem. We have the following result.

Theorem 6. ***(CDR Rule for the General Problem)*** *For any pair of jobs i and j, there exists a constant $c_{i,j}$ such that*

$$\frac{s_i'(\theta_i^*(t),t)}{s_j'(\theta_j^*(t),t)} = c_{i,j}, \tag{33}$$

for all t such that $\theta_i^*(t) > 0$ and $\theta_j^*(t) > 0$.

The proof follows directly from the same contradiction argument used in Theorem 1. This result demonstrates that the CDR Rule is a very general structural property for parallel scheduling and can greatly reduce the search space of optimal schedules.

7.3 GWF and SmartFill for the General Problem

In contrast to the CDR Rule, the GWF and SmartFill algorithms cannot be directly extended to the general problem, because they rely on knowledge of the optimal job completion order. For the baseline OPT problem, the order follows the Shortest Job First (SJF) principle. However, in the general setting, the optimal order is not known in advance. Consequently, the general scheduling problem remains open. We believe there are substantial opportunities for future research in extending the CDR Rule (and potentially GWF) to develop new algorithms for more general parallel scheduling scenarios.

8 CONCLUSION

In this paper, we solved the open problem of scheduling parallel jobs under general concave speedup functions, a fundamental challenge in modern computing systems. We established the CDR Rule, which characterizes the structure of any optimal schedule and dramatically reduces the search space. Building on this property, we developed the GWF algorithm to compute allocations under fixed derivative ratios, and combined these insights into the SmartFill algorithm. Unlike heSRPT, which allocates resources to all active jobs, SmartFill intelligently determines which jobs should receive resources and how much to allocate. We showed that for a broad class of *regular* speedup functions, SmartFill produces closed-form optimal solutions, while for non-regular concave functions it provides numerical solutions. Numerical evaluations confirmed that SmartFill consistently outperforms heSRPT across various concave speedup functions. Finally, we demonstrated that the CDR Rule generalizes to settings with heterogeneous speedup functions, time-varying resources, and broader system objectives, highlighting its potential as a unifying principle in parallel scheduling.

A A PROOF OF PROPOSITION 6

We prove Proposition 6 by induction. On one hand, it is apparent that Proposition 6 holds for $M = 1$.

On the other hand, assume Proposition 6 holds for $M = k$ ($k \geq 1$). Then consider the case $M = k+1$. Consider the time instance T_M when job M completes. Since Proposition 6 holds for $M = k$, the optimal objective after $t = T_M$, denoted by $J_{[T_M,\infty)}$, can be written as

$$J_{[T_M,\infty)} = \sum_{i=1}^{k} \alpha_i x_i', \tag{34}$$

where x_i' is the remaining size for job i at $t = T_M$. By Corollary 2.1, under the optimal scheduler, the parameters $c_1, c_2, \cdots, c_k$ exist, and their values can determined based on the scheduling results after $t = T_M$. Then for time interval $[0, T_M]$, we want to find the optimal service rate θ_M^M rate for job $M = k+1$. When the service rate for job $M = k+1$ is $\theta_M^M = \mu$, by Theorem 3, the service rate for job i ($i \leq k$) is given by

$$\theta_i^M = \mathrm{CAP}_i(B - \mu, c_1, c_2, \cdots, c_k). \tag{35}$$

We also have

$$T_M = \frac{x_{i+1}}{s(\mu)}. \tag{36}$$

The objective J in time interval $[0, T_M]$, denoted by $J_{[0,T_M]}$, can be written as

$$J_{[0,T_M]} = \sum_{i=1}^{k+1} w_i T_M = \sum_{i=1}^{k+1} w_i \cdot \frac{x_{k+1}}{s(\mu)} \tag{37}$$

For each $i = 1, \cdots, k$, we have

$$x_i' = x_i - s(\theta_i^M) \cdot \frac{x_{k+1}}{s(\mu)}. \tag{38}$$

The objective J is given by

$$J = J_{[0,T_M]} + J_{[T_M,\infty)} = \sum_{i=1}^{k+1} w_i \cdot \frac{x_{k+1}}{s(\mu)} + \sum_{i=1}^{k} \alpha_i (x_i - s(\theta_i^M) \cdot \frac{x_{k+1}}{s(\mu)}). \tag{39}$$

To minimize J, the optimal solution of μ is given by the following problem:

$$\arg\min_{\mu} \ (\sum_{i=1}^{k+1} w_i) \cdot \frac{x_{k+1}}{s(\mu)} - \Big(\sum_{i=1}^{k} a_i s(\mathrm{CAP}_i(B-\mu, c_1, c_2, \cdots, c_k)) \Big) \cdot \frac{x_{k+1}}{s(\mu)}.$$

It is easy to see that the value of μ is independent from the value of x_{k+1}. So J^* is also linear with x_{k+1}. This completes the proof. ∎

B A PROOF OF THEOREM 5

We prove the optimality of SmartFill by induction on the number of jobs. In fact, we establish the following stronger induction claim: for

every instance with k jobs, SmartFill computes an optimal scheduling matrix, and the corresponding optimal objective can be written as

$$J_k^*(x_1, \dots, x_k) = \sum_{i=1}^{k} a_i x_i, \tag{40}$$

where the coefficients a_i depend only on B, $s(\cdot)$, and the weights w_i, but not on the job sizes.

For the base case $k = 1$, allocating the entire resource B to the only job is clearly optimal. Its completion time is $x_1/s(B)$, and hence

$$J_1^* = \frac{w_1}{s(B)} x_1.$$

This is precisely the initialization of SmartFill,

$$\theta_1^1 = B, \qquad c_1 = 1, \qquad a_1 = \frac{w_1}{s(B)}.$$

Suppose that the claim holds for every instance with k jobs. We now consider an instance with $k + 1$ jobs. By Proposition 5, every optimal policy completes the jobs in the order

$$k + 1, k, \dots, 1.$$

Let

$$\tau = T_{k+1}^*$$

denote the completion time of Job $k + 1$. By Proposition 4, the allocation is constant over $[0, \tau)$ because no job completes in the interior of this interval. Denote these constant allocations by

$$\theta_i^{k+1}, \qquad i = 1, \dots, k + 1.$$

Let

$$\mu = \theta_{k+1}^{k+1}$$

be the resource allocated to Job $k + 1$ during this interval. Since Job $k + 1$ completes at time τ, we have

$$\tau = \frac{x_{k+1}}{s(\mu)}. \tag{41}$$

Moreover, full resource utilization gives

$$\sum_{i=1}^{k} \theta_i^{k+1} = B - \mu.$$

After time τ, Jobs $1, \dots, k$ form a residual k-job instance, with remaining sizes

$$x_i' = x_i - s(\theta_i^{k+1})\tau, \qquad i = 1, \dots, k. \tag{42}$$

The restriction of an optimal $(k+1)$-job schedule to $[\tau, \infty)$ must be optimal for this residual instance; otherwise, replacing its tail by a better schedule would strictly decrease the objective of the original instance.

By the induction hypothesis, SmartFill optimally schedules this residual instance, and its objective measured relative to time τ is

$$J_{\text{tail}}^* = \sum_{i=1}^{k} a_i x_i'. \tag{43}$$

The constants $c_1, \dots, c_k$ determined in the first k iterations characterize the derivative ratios of this optimal tail schedule.

The CDR Rule requires these same derivative ratios to be preserved during $[0, \tau)$ whenever the corresponding allocations are positive, while Theorem 2 gives the associated boundary conditions when some allocations are zero. Therefore, for a fixed value of μ, the allocations to Jobs $1, \dots, k$ must solve

$$\mathrm{CAP}(B - \mu, c_1, \dots, c_k).$$

By Theorem 3, this CAP instance has a unique solution, which is computed by GWF. Consequently,

$$\theta_i^{k+1} = \mathrm{CAP}_i(B - \mu, c_1, \dots, c_k), \qquad i = 1, \dots, k. \tag{44}$$

Thus, once μ is fixed, SmartFill computes the unique allocation that can be part of an optimal schedule.

It remains to optimize μ. Because every job remains unfinished throughout $[0, \tau)$, this interval contributes

$$\left(\sum_{i=1}^{k+1} w_i \right) \tau$$

to the weighted sum of completion times. Combining this contribution with (43), the objective corresponding to a fixed μ is

$$\begin{aligned} J_{k+1}(\mu) &= \left(\sum_{i=1}^{k+1} w_i \right) \tau + \sum_{i=1}^{k} a_i x_i' \\ &= \sum_{i=1}^{k} a_i x_i + \left[\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\theta_i^{k+1}) \right] \tau \\ &= \sum_{i=1}^{k} a_i x_i + \frac{\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\mathrm{CAP}_i(B - \mu, c_1, \dots, c_k))}{s(\mu)} x_{k+1}, \end{aligned} \tag{45}$$

where the last equality follows from (41) and (44).

The first term in (45) is independent of μ. Hence, the optimal allocation to Job $k + 1$ is

$$\theta_{k+1}^{k+1} \in \operatorname*{arg\,min}_{0 < \mu \le B} \frac{\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\mathrm{CAP}_i(B - \mu, c_1, \dots, c_k))}{s(\mu)}. \tag{46}$$

This is exactly the optimization performed in Eq. (26), with the minimization convention noted above. Equation (27) then computes the corresponding allocations to Jobs $1, \dots, k$ through GWF.

SmartFill next defines

$$c_{k+1} = \frac{s'(\theta_{k+1}^{k+1})}{s'(\theta_k^{k+1})} c_k. \tag{47}$$

For every $i \le k$ with a positive allocation, the existing CAP solution satisfies

$$\frac{s'(\theta_k^{k+1})}{s'(\theta_i^{k+1})} = \frac{c_k}{c_i}.$$

Together with (47), this gives

$$\frac{s'(\theta_{k+1}^{k+1})}{s'(\theta_i^{k+1})} = \frac{c_{k+1}}{c_i}.$$

Thus, Eq. (28) records the correct CDR constant for Job $k + 1$, including the corresponding boundary condition when an allocation is zero.

Finally, define

$$a_{k+1} = \frac{\sum_{i=1}^{k+1} w_i - \sum_{i=1}^{k} a_i s(\theta_i^{k+1})}{s(\theta_{k+1}^{k+1})}. \tag{48}$$

Substituting the minimizing allocation into (45) yields

$$J_{k+1}^* = \sum_{i=1}^{k} a_i x_i + a_{k+1} x_{k+1}.$$

Therefore, SmartFill computes an optimal scheduling matrix for the $(k+1)$-job instance and preserves the linear representation of the optimal objective required by the induction hypothesis.

By induction, SmartFill computes an optimal solution to OPT for every number of jobs M. ∎

REFERENCES

[1] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in computational physics*, 15(2):285–329, 2014.

[2] Michael Collier and Robin Shahan. *Microsoft azure essentials-fundamentals of azure*. Microsoft Press, 2015.

[3] Sajee Mathew and J Varia. Overview of amazon web services. *Amazon Whitepapers*, 105(1):22, 2014.

[4] Ekaba Bisong. An overview of google cloud platform services. *Building Machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 7–10, 2019.

[5] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

[8] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *2010 18th Euromicro conference on parallel, distributed and network-based processing*, pages 317–324. IEEE, 2010.

[9] Yunxiang Hu, Yuhao Liu, and Zhuoyuan Liu. A survey on convolutional neural network accelerators: Gpu, fpga and asic. In *2022 14th International Conference on Computer Research and Development (ICCRD)*, pages 100–107. IEEE, 2022.

[10] Leyuan Wang, Zhi Chen, Yizhi Liu, Yao Wang, Lianmin Zheng, Mu Li, and Yida Wang. A unified optimization approach for cnn model inference on integrated gpus. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.

[11] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[12] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in gpu datacenters: A survey. *ACM Computing Surveys*, 56(6):1–38, 2024.

[13] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18, 2003.

[14] Linus E Schrage and Louis W Miller. The queue m/g/1 with the shortest remaining processing time discipline. *Operations Research*, 14(4):670–684, 1966.

[15] Samira Ghanbarian, Arpan Mukhopadhyay, Ravi R Mazumdar, and Fabrice M Guillemin. On optimal server allocation for moldable jobs with concave speed-up. In *Proceedings of the Twenty-fifth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 191–200, 2024.

[16] Mor Harchol-Balter. Open problems in queueing theory inspired by datacenter computing. *Queueing Systems*, 97(1):3–37, 2021.

[17] Benjamin Berg, Rein Vesilo, and Mor Harchol-Balter. hesrpt: Parallel scheduling to minimize mean slowdown. *Performance Evaluation*, 144:102147, 2020.

[18] David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge university press, 2005.

[19] Suman Khakurel, Christopher Leung, and Tho Le-Ngoc. A generalized water-filling algorithm with linear complexity and finite convergence time. *IEEE Wireless Communications Letters*, 3(2):225–228, 2014.

[20] Chengwen Xing, Yindi Jing, Shuai Wang, Shaodan Ma, and H Vincent Poor. New viewpoint and algorithms for water-filling solutions in wireless communications. *IEEE Transactions on Signal Processing*, 68:1618–1634, 2020.

[21] Mor Harchol-Balter, Karl Sigman, and Adam Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Performance Evaluation*, 49(1-4):241–256, 2002.

[22] Krzysztof Sikorski. Bisection is optimal. *Numerische Mathematik*, 40(1):111–117, 1982.