Forget the Deadline: Scheduling Interactive Applications in Data Centers

Yousi Zheng*, Bo Ji†, Ness B. Shroff*‡, Prasun Sinha‡
*Dept. of ECE, The Ohio State University, Columbus, OH 43210
‡Dept. of CSE, The Ohio State University, Columbus, OH 43210
†Dept of CIS, Temple University, Philadelphia, PA 19122

***Abstract*—Many interactive applications running in data centers such as web search, social networks, online gaming, and financial services are delay-sensitive, and often have a deadline. These deadlines vary across users and applications, which makes the job scheduling problem very challenging when the overall system performance needs to be optimized. In this paper, the performance of interest is the total utility gain of multiple interactive jobs, and our objective is to maximize the total utility gain. The interactive jobs arrive to the system over time, and are allowed to be partially executed before their deadlines. We focus on the preemptive scenario, where a job in service can be interrupted by other jobs, and its service can be resumed later. We propose a deadline agnostic scheduler, called ISPEED (Interactive Services with Partial ExEcution and Deadlines). Being deadline agnostic is an attractive property of ISPEED because data center schedulers are often not privy to individual job deadlines, and thus schedulers that are deadline dependent may not be amenable to implementation. We first prove that ISPEED achieves the maximum total utility when jobs have homogeneous deadlines and their utility functions are non-decreasing and concave. Then, in the case of heterogeneous job deadlines we prove that ISPEED achieves a competitive ratio of $2+\alpha$, where α is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are concave, ISPEED has a competitive ratio of 2, while no causal scheduler can achieve a competitive ratio smaller than $\frac{\sqrt{5}+1}{2}$. Finally, we show through trace-driven simulations that ISPEED outperforms the state-of-the-art schedulers in a wide range of scenarios.**

I. Introduction

Most interactive services such as web search, social networks, online gaming, and financial services now are heavily dependent on computations at data centers because their demands for computing resource are both high and variable. Interactive services are time-sensitive as users expect to receive a complete or possibly partial response within a short period of time. The exact nature of the quality of the response and the allowed time-limit for completing a job may vary across individual users and applications, but interactive jobs typically have the following characteristics:

1) **Partial Execution:** Although fully completed jobs are preferred, partially completed jobs are also acceptable to the end user. For example, in a web search, users mostly care about the top few search results which can be obtained without completing the full job.
2) **Time-Sensitive:** Each job has its corresponding strict deadline, whose exact value may be difficult to determine a priori. For example, in a web search, users may not want to wait beyond a few seconds.
3) **Heterogeneity:** Various characteristics of the jobs such as their utility functions and their deadlines may all be heterogeneous. Such heterogeneity is inherent in data centers as they support a variety of users (e.g., VIP customers vs. free users) and applications.
4) **Concavity:** Based on the the law of diminishing marginal utility (Gossen's laws), most utility functions are concave. Studies on the Bing search engine [7], [8] have mapped this utility function and it has been found to be concave on average. We have also made similar observations in our own experiments (see Section VI). However, the utility functions are not necessarily concave for all the jobs.

The comparison between the state-of-the-art schedulers and our proposed ISPEED (Interactive Services with Partial ExEcution and Deadlines) scheduler is shown in Table I. In this table, "Yes" means that related issues are considered in the literature, although the exact definitions and approaches could be different from our paper. More detailed discussion about the related work is provided in Section VII.

Scheduler	Partial Execution	Time-sensitive	Hetero-geneity	Concave and Non-concave
[3], [4], [13] [15]–[17]	-	Yes	-	-
[1], [11], [12]	-	-	Yes	-
[7], [8]	Yes	Yes	-	Concave
[2], [5]	Yes	Yes	Yes	-
[9], [10]	-	Yes	Yes	-
ISPEED	Yes	Yes	Yes	Both

TABLE I
Comparison between State-of-the-art Schedulers and ISPEED

In this paper, we study the problem of scheduling interactive jobs in a data center with the goal of maximizing the total utility of all jobs. This problem is particularly challenging because future arrivals of jobs and their requirements are unknown, which renders it difficult to make the right scheduling decisions for jobs currently in the system. We assume that service preemption is allowed, and summarize our contributions as follows:

- First, we propose a deadline agnostic scheduler, called ISPEED (Section II), and prove that ISPEED maximizes

the total utility when the jobs have homogeneous deadlines and their utility functions are non-decreasing and concave (Section III).

- Second, when the jobs have heterogeneous deadlines, we prove that IPSEED achieves a competitive ratio of $2+\alpha$, where α is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are non-decreasing and concave, the competitive ratio of ISPEED can be improved to 2 (Section IV).
- We discuss various challenges in practical implementations, and show the robustness of our proposed solution in dealing with these issues (Section V). Finally, we conduct extensive trace-driven simulations, and show that ISPEED outperforms three other widely used schedulers (Section VI).

The design of our solution is grounded on a rigorous mathematical foundation. Since our solution does not require the knowledge of deadlines, it is suitable for real data centers. Indeed, in practice it is non-trivial to obtain the exact information about individual user deadlines. The performance gap between our solution and the best causal scheduler is small in terms of the competitive ratio (2 vs. $\frac{\sqrt{5}+1}{2}$), which shows that there is a limited space for further improvement upon our solution. We also show how this basic solution based on theoretical foundations can be enhanced to work well in real settings when a number of practical issues need to be addressed.

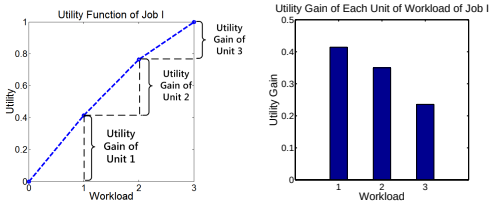
The remainder of the paper is organized as follows. In Section II, we describe the system model that we consider for our analytical results, and propose our ISPEED scheduler. In Sections III and IV, we analyze the performance of ISPEED in the scenarios where the job deadlines are homogeneous and heterogeneous, respectively. Performance evaluations based on simulations are provided in Section VI. Finally, we discuss the related work in Section VII, and conclude our paper in Section VIII.

II. System Model and ISPEED Scheduler

In this section, we describe the system model that we consider for our analytical results, and propose our scheduler ISPEED. In Section V, we will discuss various practical issues and how our proposed solution can be adapted to deal with these issues.

A. System Model

Consider a data center with N machines. Let there be n jobs arriving into the system within a specific period of time. Let W_i denote the total workload of job i, and let $U_i(x)$ denote the utility function for job i, where the argument x denotes the amount of completed workload for job i. We assume that each machine can process one unit of workload in each time slot. We define the marginal utility gain as the increase in the total utility when one additional unit of workload is served. In Fig. 1, we provide an example of the utility function and the marginal utility gain for each workload. Let $w_{i,t}$ denote



(a) A utility function (b) Marginal utility gain

Fig. 1. Example Utility Function

the amount of workload scheduled for job i in time slot t. Clearly, $w_{i,t}$ is also the number of machines allocated to job i. Therefore, we must have $\sum_{i=1}^{n} w_{i,t} \leq N$ and $w_{i,t} \geq 0$ for all t. For each job i, there is an associated arrival time a_i and a deadline d_i. The workload of the job cannot be scheduled after the deadline. In this paper, we focus on the preemptive scenario, where a job that is being served can be interrupted by other jobs, and its service can be resumed later.

Let $w_i \triangleq \sum_{t=a_i}^{d_i} w_{i,t}$ be the amount of workload that job i completes before its deadline. Clearly, we have $w_i \leq W_i$ for all $i \in \{1, 2, \ldots, n\}$. Then, the utility maximization problem we consider in this paper can be formulated as follows.

$$\begin{aligned} \max_{w_{i,t}} \ & \sum_{i=1}^{n} U_i(w_i) \\ s.t. \ & \sum_{i=1}^{n} w_{i,t} \leq N, \ w_{i,t} \geq 0, \ \forall t, \\ & \sum_{t=a_i}^{d_i} w_{i,t} = w_i \leq W_i, \ \forall i \in \{1, ..., n\} \\ & w_{i,t} = 0 \text{ for } t < a_i \text{ and } t > d_i, \ \forall i \in \{1, \cdots, n\}. \end{aligned} \tag{1}$$

In the next section we consider a simplified version of the above model in which jobs are constrained by homogeneous deadlines, i.e., all the deadlines are the same. In Section IV, we study the above problem (with heterogeneous deadlines for each job).

B. ISPEED scheduler

We present a deadline agnostic scheduler, called **ISPEED Scheduler:** In each time-slot, repeatedly schedule the first unit of workload from the list of waiting jobs that have the highest marginal utility gain in this time slot, until there is no machine available to schedule, or no waiting job. Observe that multiple units of workload of a single job can be scheduled within a time slot. Upon arrival, a job is added to the queue of waiting jobs and is deleted when it is completely served or its deadline expires. The detailed operations of ISPEED are described in Algorithm 1.

III. Performance Analysis of ISPEED with Homogeneous Deadlines

We show that ISPEED is an optimal solution in the scenario when the utility functions are non-decreasing and concave, and there are homogeneous deadlines, i.e., deadlines of all jobs are the same. Note that if there is no deadline for jobs,

Algorithm 1 Interactive Services with Partial ExEcution and Deadlines (**ISPEED)**

Input: List of unfinished jobs (including new arrivals in the current time slot) J whose deadlines have not expired in each time slot.
Output: Scheduled machines for jobs, and updated remaining jobs list J in each time slot.

Sort J in a non-increasing order of the marginal utility gains;
$d \leftarrow N$; // d is the number of available machines; N is the total number of machines.
while J is not empty and $d \neq 0$ // There are available machines **do**
    i^* is the first job in J;
    Allocate a machine to the first unit of workload of Job i^*;
    Delete the allocated unit of workload from i^*;
    if All the workload of Job i^* is scheduled **then**
        Delete Job i^* from J;
    else
        Update Job i^* in J corresponding to its new marginal utility gain;
    end if
    $d \leftarrow d - 1$;
end while
for Each Job i in J **do**
    if Job i will expire by the end of this time slot **then**
        Delete Job i from J;
    end if
end for

but a stopping time for the system, then it is equivalent to homogeneous deadlines for all the jobs. Later in the next section, we show that ISPEED can achieve a competitive ratio of $2 + \alpha$ for non-decreasing utility functions while no other causal scheduler can achieve a competitive ratio less than $\frac{\sqrt{5}+1}{2}$. Also note that our proposed ISPEED scheduler is deadline agnostic, i.e., it makes scheduling decisions without having the information of job deadlines.

For a schedule S, let $S(t)$ be the schedule in time slot t (a list of units of workload), and let $S(t,p)$ be the unit of workload in time slot t and position p within the list.

The proof proceeds as follows. We start with an optimal (and feasible) schedule and at each step the schedule is altered to another feasible schedule which matches one more element of the ISPEED schedule without reducing the total utility gain. By repeating the above operations, we end up with a schedule that matches with the ISPEED schedule and has a total utility gain that is no smaller than the total utility gain of the optimal schedule. This implies that ISPEED also produces an optimal solution.

Theorem 1: Under the assumptions that the utility functions are non-decreasing and concave, and that the jobs have homogeneous deadlines, ISPEED achieves the optimal total utility.

Proof: Let O be an optimal schedule with total utility $U(O)$ and let G be the schedule computed by ISPEED with total utility $U(G)$. Let τ be the first time-slot when the two schedules differ. For both O and G, within each time-slot we sort the units of workload in a non-decreasing order according to their utility gains, respectively. While going through the two sorted lists in time-slot τ in both schedules, we arrive at the first position (say p) where there is a difference. It implies that the first $p-1$ positions have the exact same units of workload in both schedules, and at least one of the two schedules (O and G) has more than $p-1$ units. Now, there are three cases:

1) **Both $O(\tau)$ and $G(\tau)$ have more than $p-1$ units:** Let $o = O(\tau,p)$ and $g = G(\tau,p)$ be the workload scheduled in position p in time slot τ under schedules O and G, respectively. We know that $o \neq g$. Also, it is easy to see that the utility gain of g is no smaller than that of o, by the concavity of the utility functions and the greedy operations of ISPEED. Suppose that there exists a position (τ', p') beyond (τ, p) (a later position in slot τ or a position in a later slot) in which the same unit g is scheduled in O. Then, we obtain another schedule O' from O by swapping o and g. This swapping operation is feasible due to the following reason: 1) clearly it is feasible to move o to a later position than (τ,p) in O', because all the jobs have the same deadlines; 2) it is feasible to schedule g in position (τ,p) in O' as O and G have the same schedule by position $(\tau, p-1)$ and g is scheduled in position (τ,p) in G. As a result, O' matches one more position with G than O does, while keeping the total utility unchanged.
   Alternately, if g is not scheduled in O beyond position (τ,p), then we obtain O' from O by replacing the unit in position (τ,p) with g. This operation is feasible as O and G have the same schedule by position $(\tau, p-1)$ and g is scheduled in position (τ,p) in G (by ISPEED). Further, this operation does not reduce the total utility as the utility of g is no smaller than that of o. So O' will have a total utility no smaller than that of O and will match with G in one more position.
2) **$O(\tau)$ has $p-1$ units and $G(\tau)$ has more than $p-1$ units:** Consider job $g = G(\tau,p)$. The number of machines in the system is at least p. We obtain O' from O by scheduling unit g in position p, i.e., $O'(\tau,p) = g$. If g is not scheduled in slots after τ in O, then this operation does not reduce the total utility of O. Hence, we have $U(O') \geq U(O)$.
   If g is scheduled in a slot after τ in O, say $g = O(\tau', p')$, then we obtain O' from O by switching unit g from position (τ', p') to position (τ, p). This operation is feasible as O and G have the same schedule by position $(\tau, p-1)$ and g is scheduled in position (τ,p) in G (by ISPEED). It is also easy to see that this operation keeps the total utility unchanged, i.e., $U(O') = U(O)$, and that the new schedule O' matches with G in one more position.
3) **$O(\tau)$ has more than $p-1$ units and $G(\tau)$ has $p-1$ units:** We want to show that this case does not occur. Consider the workload $o = O(\tau,p)$. Then, o must also be available for scheduling under ISPEED in time slot τ, and thus in position (τ,p) ISPEED must schedule workload o or another workload with a larger marginal utility gain than o. This contradicts with the assumption that $G(\tau)$ has $p-1$ units, which further implies that

case 3) does not occur.

By repeating the above steps, we reach a final schedule that exactly matches with G. Since the total utility does not decrease at any step, we end up with a schedule whose total utility is no smaller than that of O. This implies that $U(G) \geq U(O)$. Due to the assumed optimality of O, we have $U(G) = U(O)$, i.e., G is also an optimal solution for Eq. (1) with the assumptions of homogeneous deadlines and non-decreasing and concave utility functions. ■

IV. Performance Analysis of ISPEED with Heterogenous Deadlines

In this section, we consider a more general scenario of heterogeneous job deadlines. It is known that no causal scheduler can achieve a competitive ratio less than $\frac{\sqrt{5}+1}{2}$ [6]. Then, we prove that ISPEED has a competitive ratio of $2+\alpha$, where α is a shape parameter for a large class of non-decreasing utility functions, and this competitive ratio can be improved to 2 when the utility functions are non-decreasing and concave.

We now give a standard definition of the competitive ratio. A scheduler S has a **competitive ratio c**, if for any arrival pattern, the total utility $U(S)$ of S satisfies that

$$\frac{U^*}{U(S)} \leq c, \tag{2}$$

where U^* is the maximal total utility over all possible schedulers (including non-causal schedulers) for the same sample path.

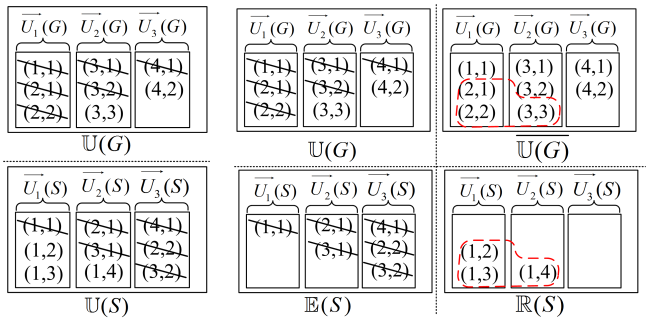
It has been shown in [6] that the lower bound on the competitive ratio for all causal schedulers is $\frac{\sqrt{5}+1}{2} \approx 1.618$, which considers a single-server system, but the counterexample also applies in our system. The main idea of the proof is as follows. For any given causal scheduler, one can always construct a special job arrival pattern such that the scheduler cannot achieve a total utility that is larger than $\frac{\sqrt{5}-1}{2}$-fraction of the maximal achievable total utility. Next, we prove an achievable competitive ratio of ISPEED for a large class of non-decreasing utility functions.

Theorem 2: Assume that the utility functions are non-decreasing for all jobs. Suppose that there exists a constant $\alpha \geq 0$, such that the utility function U_i of any job i satisfies

$$\frac{U_i(m) - U_i(k)}{m-k} \leq (1+\alpha)\nabla U_i(k), \forall k,\ \forall m > k, \tag{3}$$

where $\nabla U_i(k) = U_i(k) - U_i(k-1)$ is the marginal utility gain of the k^{th} unit of workload of job i. Then ISPEED has a competitive ratio of $2+\alpha$.

Proof: We can describe a scheduler S as follows. For each time slot t, the schedule can be represented as a set $\overrightarrow{U}_t(S) = \{(i,j)\}$, if the j^{th} unit of workload of job i is scheduled in time slot t. The marginal utility gain of (i,j) is $U_{i,j}$. Then, the schedule can be represented by a set $\mathbb{U}(S) = \bigcup_t \overrightarrow{U}_t(S)$. Let $\mathbb{U}(G)$ be the representation of the schedule under the same arrival pattern for ISPEED. We make a copy of $\mathbb{U}(G)$ and denote it by $\overline{\mathbb{U}(G)}$. We want to show that the summed utility gain of both $\mathbb{U}(G)$ and $\overline{\mathbb{U}(G)}$ times $1+\alpha$ is greater than or equal to the summed utility gain of $\mathbb{U}(S)$ for an arbitrary scheduler S. Our proof has two steps.



(a) The Erasing Method (b) Construction for Bound of $2+\alpha$

Fig. 2. An example to illustrate the proof of Theorem 2

In the first step, we compare $\mathbb{U}(G)$ with $\mathbb{U}(S)$, and remove the units of workload with the same index (i,j). In other words, if some units of jobs are scheduled by both scheduler S and ISPEED, we pick them out and include them in a set called $\mathbb{E}(S)$. We erase all such elements from both $\mathbb{U}(G)$ and $\mathbb{U}(S)$. This erasing process is illustrated in Fig. 2(a). Since $\mathbb{E}(S) \subseteq \mathbb{U}(G)$, we get

$$\sum_{(i,j)\in\mathbb{E}(S)} U_{i,j} \leq \sum_{(i,j)\in\mathbb{U}(G)} U_{i,j}. \tag{4}$$

In the second step, we consider all the remaining elements in $\mathbb{U}(S)$ after the erasing process. Let $\mathbb{R}(S)$ denote the set of the remaining elements, and we have $\mathbb{R}(S) = \mathbb{U}(S) \backslash \mathbb{E}(S)$. For the purpose of illustration, we provide an example of set $\mathbb{E}(S)$ and $\mathbb{R}(S)$ in Fig. 2(b). Due to the way we construct set $\mathbb{R}(S)$, we have $\mathbb{R}(S) \cap \overline{\mathbb{U}(G)} = \emptyset$.

To compare $\mathbb{R}(S)$ with $\overline{\mathbb{U}(G)}$, we first show that the scheduled number of units in $\overline{\mathbb{U}(G)}$ is no less than that in $\mathbb{R}(S)$ in each time slot. Then, we group all the units scheduled in $\mathbb{R}(S)$ into disjoint pools, each of which corresponds to a different job. For each pool in $\mathbb{R}(S)$, we also construct a corresponding pool in $\overline{\mathbb{U}(G)}$. For each pair of such pools, we show that the marginal utility gain of each unit from the pool in $\overline{\mathbb{U}(G)}$ is no less than that of the first unit from the pool in $\mathbb{R}(S)$. Finally, we obtain our results by summing the utility gain over all the pools for $\mathbb{U}(G)$ and $\mathbb{R}(S)$, respectively, and comparing their summed utility.

First, we want to show that in each time slot, the number of scheduled units in $\overline{\mathbb{U}(G)}$ is no less than that in $\mathbb{R}(S)$. Let $N_G(t)$ and $N_R(t)$ be the units of workload in $\overline{\mathbb{U}(G)}$ and $\mathbb{R}(S)$ in time slot t, respectively. If $N_G(t) = N$, then clearly we have $N_R(t) \leq N = N_G(t)$. If $N_G(t) < N$, then we must have $N_R(t) = 0$. The reason is as follows. Suppose $N_R(t) > 0$. This means that there exists some units of a job, say u, which belongs to $\mathbb{R}(S)$. Then, the first unit of u in $\mathbb{R}(S)$ must also belong to $\overline{\mathbb{U}(G)}$ because the unit of job u is feasible for ISPEED to schedule and ISPEED has idle machines (i.e., $N_G(t) < N$). However, this contradicts with the fact that $\mathbb{R}(S) \cap \overline{\mathbb{U}(G)} = \emptyset$.

Let's assume that the workload in $\mathbb{R}(S)$ belongs to K jobs. Among these K jobs, each job has N_1,...,N_K units

of workload scheduled, respectively. We consider the first scheduled unit of each job in $\mathbb{R}(S)$, whose utility gain is denoted by F_1,...,F_K. Then, we obtain the following:

$$\begin{aligned}\sum_{(i,j)\in\mathbb{R}(S)} U_{i,j} &= \sum_{k=1}^{K} \text{Utility gain of the } k^{th} \text{ job in } \mathbb{R}(S) \\ &\leq (1+\alpha)\sum_{k=1}^{K} N_k F_k,\end{aligned} \tag{5}$$

where the inequality is from our assumption of Eq. (3).

Now, we construct K disjoint pools of units in $\overline{\mathbb{U}(G)}$ corresponding to the location of jobs in $\mathbb{R}(S)$. In time slot t, if there are some units of job k that are scheduled in $\mathbb{R}(S)$, then we choose the same number of units in $\overline{\mathbb{U}(G)}$, and put them into the k^{th} pool of $\overline{\mathbb{U}(G)}$. We provide an example in Fig. 2(b), where 2 units and 1 unit of Job 1 are scheduled in $\mathbb{R}(S)$ in time slot 1 and 2, respectively. Then, we arbitrarily choose 2 units and 1 unit in $\overline{\mathbb{U}(G)}$ in time slot 1 and 2, respectively, and put them in the first pool of $\overline{\mathbb{U}(G)}$. Since $N_G(t) \geq N_R(t)$ holds for any time slot t, it is feasible to construct the K disjoint pools.

Let H_k be the minimal marginal utility gain in the k^{th} pool of $\overline{U(G)}$. Then, we have $H_k \geq F_k$, $\forall k \in 1,2,...K$. This is because 1) ISPEED always chooses the job whose first unscheduled unit has the largest utility gain, and 2) the first unit of each job in $\mathbb{R}(S)$ is feasible but does not belong to $\overline{\mathbb{U}(G)}$. Then, from Eq. (5) we have

$$\begin{aligned}&\sum_{(i,j)\in\mathbb{R}(S)} U_{i,j} \leq (1+\alpha)\sum_{k=1}^{K} N_k F_k \\ &\leq (1+\alpha)\sum_{k=1}^{K} N_k H_k \leq (1+\alpha)\sum_{(i,j)\in\overline{\mathbb{U}(G)}} U_{i,j}.\end{aligned} \tag{6}$$

By summing Eq. (4) and Eq. (6), we obtain

$$\sum_{(i,j)\in\mathbb{U}(S)} U_{i,j} \leq (2+\alpha)\sum_{(i,j)\in\mathbb{U}(G)} U_{i,j}, \tag{7}$$

which is equivalent to $U(S) \leq (2+\alpha)U(G)$. ■

In Theorem 2, α can be viewed as a shape parameter, which is determined by the shape of the utility functions. In particular, we have $\alpha = 0$ when the utility functions are non-decreasing and concave. Then, we can have the following theorem.

Theorem 3: If all the utility functions are non-decreasing and concave, then 2 is a competitive ratio of ISPEED, and no competitive ratio smaller than 2 can be achieved by ISPEED.

Proof: Theorem 3 contains two parts: I) ISPEED has a competitive ratio 2; II) Any constant which is less than 2 is not a competitive ratio for ISPEED.

Since all the utility functions are non-decreasing and concave, Eq. (3) is satisfied when $\alpha = 0$. By Theorem 2, we can directly get that 2 is a competitive ratio of ISPEED. The first part is proven.

To prove the second part, we only need to give a counter example to show that the competitive ratio cannot be less than 2. We construct an arrival pattern to show that the competitive ratio of ISPEED cannot be less than 2. Let us consider two arrivals in a data center with 1 machine. Let Job I arrive in time slot 1 with utility gain c ($0 \leq c < 1$) and deadline the time slot 1. Let Job II also arrive in time slot 1 with utility gain 1 and deadline 2.

It is easy to see that ISPEED will schedule Job II in the first time slot, and no job will be available in the following time slots. Then, the totaly utility of ISPEED is 1. However, consider a scheduler S that schedules Job I in the first time slot, and Job II in the second time slot, then the total utility of S is $1+c$. Thus, the competitive ratio of ISPEED is no less than $1+c$. Since c can be arbitrarily close to 1, the competitive ratio of ISPEED is no less than 2. ■

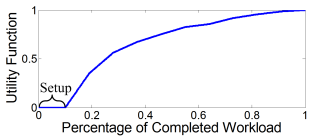
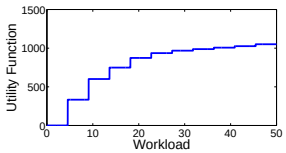


Fig. 3. Utility Function with Setup

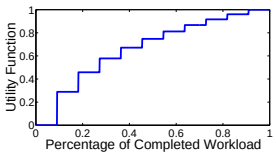
V. Practical Considerations

So far, our discussion has been focused on the model that supports partial execution, heterogeneous customers (e.g., VIP vs. free customers), and heterogeneous job deadlines. In this section, we discuss a number of other critical implementation issues in practice and how our proposed solution can be adapted to deal with these issues.

- **Initialization Cost:** Some applications need a non-negligible amount of preparation time before a job can begin execution. The preparation time is needed for various activities including building the virtual machine, collecting necessary data from remote machines and waking up physical devices from sleep. During the setup time, the job may yield minimal or zero utility gain. So, the utility function may have a shape as in Fig. 3. Since the utility gain in the setup period is small, then the shape parameter α in Section IV will be large, which renders the theoretical performance guarantees shown in Theorem 2 loose. However, if the initialization time is a small fraction of the total processing time, we still expect our solution to perform well.
- **Multiple Tasks Per Job:** Each job may contain multiple tasks and the utility of the job may increase in a discrete fashion when a task is completed. For example, MapReduce jobs typically have multiple tasks. The utility function may become a step function in such cases (Fig. 4). As the step size becomes small, the function is approximately continuous and possibly concave.
- **Cost of Parallelization and Interruption:** When a job is divided into multiple units which are executed in parallel, there is an additional cost of parallelization. In addition, when a job is interrupted, the state needs to be stored and when it is resumed, the state needs to be recovered. The costs for such actions have not been considered in

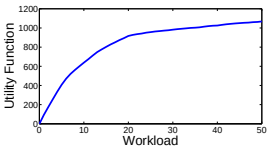


(a) Based on Collected Data in Google Search (10 Tasks Example)

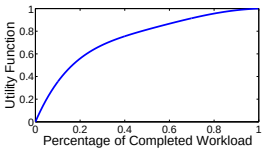


(b) Based on Data in Bing from MSR (10 Tasks Example)

Fig. 4. Utility Function with Multiple Tasks



(a) Google (Based on Collected Data from Users)



(b) Bing [7], [8]

Fig. 5. Utility Function for Search Engine

our model. Alternatively, we investigate the effect of such costs in detail in our simulations (Section VI). If the cost of parallelization and interruption is extremely large, then a job may need to run in a non-preemptive fashion.

- **Robustness to Incomplete Information:** For a job arriving to the data center, there are various important information associated with the job, such as the total workload, the deadline, the utility function, and so on. In general, it is very difficult, if not impossible, to obtain these information upon the arrival of the job. However, these information are crucial to making scheduling decisions. Our solution only requires the knowledge of the currently available jobs and their utility gains in the next time slot, and is thus inherently robust to lack of the knowledge of various information of the jobs. This is a unique strength of our solution.
- **System Overloading:** When the traffic intensity exceeds the capacity of the data center, some jobs have to be abandoned or partially completed. Allowing partial execution has already been explicitly considered in our model described in Section II. As a result, our proposed solution still works in an overloaded system.

VI. Simulation Results

A. Understanding the Nature of Utility Function

Before evaluating the performance of different schedulers, we need to understand the utility function by collecting data from users. We ask 10 users to submit 3 search jobs in the Google search engine. For each such job, the user is required to give a score to show the utility gain of each search result given by Google. Each user gives scores to at least 30 results for each job to indicate the utility gain for this job. The search keywords are chosen by the users. In total we have about 1000 collected scores from the users. Based on these scores, we create the average utility function as shown in Fig. 5(a). For the utility functions, several jobs have non-concave utility functions. However, most of the jobs are close to concave functions.

Microsoft Research (MSR) has also investigated the utility function for Microsoft's Bing search engine in [7], [8], which is shown in Fig. 5(b). Both curves are close to concave, which means that the performance guarantee is tight as α is small. Our evaluation is based on both datasets.

B. Evaluation Setup

Next, we describe the default setting of our experimental evaluations. We consider a data center with $N = 100$ machines. There are 400 jobs arriving into the data center and their arrival time slots are uniformly distributed within the first 20 time slots. Their deadlines are arrival time slot plus an integer which is either 1 (tight) or 100 (loose). The workload of each job is 50 units. There are two types of users: VIP users and free users. For the 20 VIP users, each have 100 times larger utility than a free user, for the same amount of completed workload. To model the cost of interruption and parallelization, we introduce a parameter ϵ to represent the time cost (whose unit is time slot) of such operations. We assume that ϵ-fraction of the total workload is used to account for the cost of interruption and parallelization. The default value of ϵ is 0.05. Also, there are several units of workload used to build up the environment for the job before the job can be executed. These units of workloads have zero utility gain. The default initialization cost is 5% of the total workload. Unless otherwise mentioned, these default parameters are used throughout our evaluation. These parameters are also summarized in Table II.

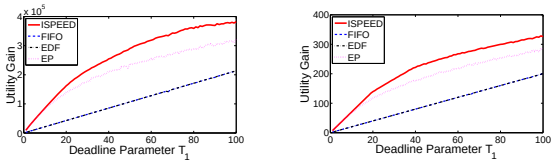
Parameter	Default Values
Number of Machines N	100
Number of Jobs	400
Number of VIP Jobs	20
Workload per Job	50
Tight Deadline from Arrivals	1 time slot
Loose Deadline from Arrivals	100 time slots
Cost Parameter ϵ	5%
Initialization Cost	5%

TABLE II
DEFAULT PARAMETERS IN OUR EVALUATIONS

We consider three other widely used schedulers and compare their performance with our proposed ISPEED: FIFO (First In First Out), EDF (Earliest Deadline First) and EP (Equal Partitioning, or called Fair Scheduler). The definitions of these schedulers are as follows:

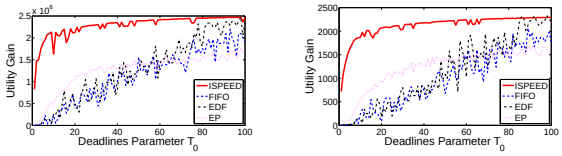
1) **FIFO (First In First Out):** Scheduler schedules the jobs corresponding to their arrival times. The jobs with earlier arrival times have a higher priority to schedule.
2) **EDF (Earliest Deadline First):** Scheduler schedules the jobs corresponding to their deadlines. The jobs with earlier deadlines have a higher priority to schedule.
3) **EP (Equal Partitioning):** Scheduler schedules the jobs with equal opportunity. All the jobs will obtain equal resources, except when the jobs do not have any available units to schedule. The EP Scheduler is a totally fair scheduler.

Different from [7], we assume that the job can be preempted while incurring some additional cost.



(a) Google (b) Bing

Fig. 6. Homogeneous Deadlines



(a) Google (b) Bing

Fig. 7. Heterogeneous Customers and Deadlines

C. Homogeneous Deadlines

We let the jobs uniformly arrive to the system from time slot 1 to time slot T_1. All the jobs have homogeneous deadlines, which is set to be T_1. To keep homogeneity among all the jobs, all users in the homogeneous scenario are free users with the same utility function. Also, to verify the optimality of ISPEED, we ignore the cost of migration and initialization in this part. Then, the performance of schedulers is summarized in Fig. 6(a) (based on the collected data from Google search) and Fig. 6(b) (based on the data from Bing from MSR).

From these two figures, we observe that the performance of utility gain of ISPEED is consistently better than that of the others in the scenario of homogeneous job deadlines, without consideration of additional cost. For example, when the deadline is 50 time slots, the performance of ISPEED is 172% better than EDF and FIFO, and is 22% better than EP in Google search engine. From Fig. 6, we can see that FIFO and EDF perform similarly, and their total utility gains increase almost linearly as the deadline T_1 increases. This is because both FIFO and EDF schedule the jobs one by one and execute the jobs completely if their deadlines have not expired yet. When the deadline increases and hence there is more time available for scheduling the jobs, the number of jobs executed by FIFO and EDF increases and thus the obtained total utility gain both increases proportionally.

We observe that the performance of ISPEED is better than EP, and both of them are much better than FIFO and EDF. This is consistent with the results given in [7]. However, we will show later that the performance of ISPEED is much better than EP, FIFO and EDF in the scenario of heterogeneous deadlines. This shows that ISPEED is more robust to uncertainty in job deadlines.

D. Heterogeneous Deadlines

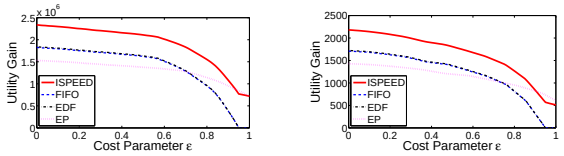
We now study the performance of different schedulers under heterogeneous deadlines. For simplicity, we introduce a parameter T_0. We let the arrival times of jobs to be uniformly distributed among the first T_0 time slots, and let their heterogeneous deadlines to be their arrival times plus a random value, which shows how urgent the job is. We choose two types of jobs: one is very urgent, whose deadline is the very next time slot after the jobs arrive; the other is not urgent, whose deadline is T_0 time slots after the jobs arrive. We introduce different types of customers (VIP vs. free users) and choose T_0 from 1 to 100. The performance of different schedulers with heterogeneous deadlines is summarized in Fig. 7(a) (Google) and Fig. 7(b) (Bing). From these two figures, we can see that the performance of ISPEED is better than the other three schedulers, especially when the deadlines are not extremely tight or loose. For example, when the parameter of deadline T_0 is 40 time slots, the performance of ISPEED is 80% better than EP, 195% better than EDF, and 135% better than FIFO in Google search engine.

From Fig. 7, we observe that EP performs much worse than ISPEED, especially when there is a large variation in the deadlines among different jobs. This is because when the deadline is very tight for some jobs, and very loose for others, EP will lose the opportunity of scheduling the newly arrived jobs, which may have an urgent deadline but large utility gains. For example, suppose that there is a job with a loose deadline, which arrived at the system before the current slot and has a small marginal utility gain in this slot. Also, there is a newly arrived job, whose deadline is very tight. Then serving the latter one will produce more utility gain than serving the former one. EP gives them the same weight to share the resource, while ISPEED prefers giving the resource to the jobs with more utility gain, and thus leads to a better performance.

E. Different Cost Parameters

We also study the impact of ϵ (cost parameter) when it changes from 0 (equivalent to no parallelization and interruption cost) to 1. When ϵ is 1, it means that the entire slot is used to prepare the machine for execution. The total utility gain of schedulers with different ϵ is shown in Fig. 8. Since FIFO and EDF greedily parallelize all the workload of a job, the total utility gain of these two schedulers are very close when ϵ is going to 1. As ϵ approaches 1, FIFO and EDF waste almost all of the resources in parallelization and interruption cost, which leads to very small utility. On the other hand, ISPEED and EP work well in the scenario with higher ϵ. EP gives equal chances to all jobs, which leads to a stable number of allocation of machines for each job. Roughly speaking, the system behaves in a non-preemptive fashion, and the effect of the cost remains low. ISPEED prefers jobs with larger marginal utility gain. From Fig. 8, we see that the performance of ISPEED is better than EP in most cases. When ϵ is close to 1, the performance of ISPEED is very close to EP. Also, the performance of ISPEED is consistently better than FIFO and EDF, for all values of ϵ. For example, when ϵ is equal to 0.3, the utility gain of ISPEED is 53% better than EP, and 29% better than FIFO and EDF for the Google dataset.

More simulation results of the heterogeneous scenario, different job initialization cost scenario, multiple tasks scenario, and fairness analysis of different schedulers can be found in our online technical report [14].



(a) Google (b) Bing

Fig. 8. Parallelization and Interruption Cost

VII. Related Work

In [3], [4], [13], [15]–[17], different schedulers for data centers under the MapReduce framework are proposed. In [3], the authors design a scheduler for minimizing the weighted sum of the job completion times, and propose approximation algorithms that work within a factor of 3 of the optimal. Based on the work of [3], the authors in [4] consider the effect of the Shuffle phase between the Map and Reduce tasks. In [15]–[17], the authors consider the problem of minimizing total flow time, where no constant competitive ratio can be achieved under some general assumptions. The authors proposed a new metric called efficiency ratio, and under this new metric proposed schedulers with provable performance guarantees. However, these schedulers consider the scenario where the entire workload needs to be completed. Partial execution is not considered in these works.

In [1], [11], [12], the problem is analyzed based on classification of QoS. QoS-adaptive systems are proposed for supporting different QoS levels in [1], [11], [12]. However, they assume homogeneous utility gain in the same QoS level or assume that the utility gain is directly determined by the current allocation regardless of previously allocated workload. Moreover, they do not consider job deadlines.

In [7], [8], both deadlines and concavity of the utility functions are studied. However, their analytical results are based on the assumption of homogeneity and strict concavity. For example, homogeneity of all the deadlines and utility functions is necessary in the proof of Lemma 1 in [7]. For heterogeneous or non-concave scenarios, no performance guarantee is given.

In [2], [5], different schedulers and corresponding performance guarantees are studied with the partial execution property. However, these works are limited to the (weighted) linear utility functions assumption. In our work, we study both concave and non-concave utility functions. Thus, the linear scenario can be viewed as a subcase of our study.

Authors in [9], [10] propose another mechanism to explore the utility maximization problem, although the original problems in their work are to study how to provide QoS to maximize utility for wireless clients. In their work, the utility is determined by stochastic parameters, e.g., the delivery ratio guarantee with probability 1. Different from the stochastic study, our work focuses on the worst-case guarantee, which is stronger and can be characterized by the competitive ratio.

VIII. Conclusion

This paper explored the problem of scheduling interactive jobs in data centers which have four distinct properties: partial execution, deadlines, heterogeneity and concavity. We developed a greedy scheduler called ISPEED, and proved that ISPEED has guaranteed worst-case performance in terms of total utility gain, which is characterized by the competitive ratio. Further, we evaluated the performance of ISPEED in various practical scenarios via trace-driven simulations, which show that ISPEED achieves a larger utility gain compared to the state-of-the-art schedulers. For future works, it would be interesting to consider a more general model that potentially accounts for multiple tasks, multiple resources, initialization cost, and parallelization and interruption cost. The key challenge will be to design an efficient and simple scheduler with provable performance guarantees.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. User-Level QoS-Adaptive Resource Management in Server End-Systems. *IEEE Transactions on Computers*, 52(5):678–685, 2003.

[2] E.-C. Chang and C. Yap. Competitive Online Scheduling with Level of Service. *Computing and Combinatorics*, 2108:453–462, 2001.

[3] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *Proceedings of IEEE Infocom*, pages 3074–3082, March 2011.

[4] F. Chen, M. Kodialam, and T. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. In *Proceedings of IEEE Infocom*, March 2012.

[5] F. Y. L. Chin and S. P. Y. Fung. Improved Competitive Algorithms for Online Scheduling with Partial Job Values. *Computing and Combinatorics*, 2697:425–434, 2003.

[6] B. Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Conference on Information Sciences and Systems, Johns Hopkins University*, pages 434–439, March 2001.

[7] Y. He, S. Elnikety, J. Larus, and C. Yan. Zeta: Scheduling interactive services with partial execution. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC'12)*, 2012.

[8] Y. He, Z. Ye, Q. Fu, and S. Elnikety. Budget-based control for interactive services with adaptive execution. In *Proceedings of the 9th ACM International Conference on Autonomic Computing (ICAC'12)*, September 2012.

[9] I.-H. Hou and P. R. Kumar. Utility maximization for delay constrained qos in wireless. In *Proceedings of IEEE International Conference on Computer Communications (Infocom'10)*, pages 1–9, March 2010.

[10] I.-H. Hou and P. R. Kumar. *Packets with Deadlines: A Framework for Real-Time Wireless Networks*. Mongan & Claypool Publishers, San Rafael, California, USA, 2013.

[11] J. Huang, P.-J. Wan, and D.-Z. Du. Criticality and QoS-Based Multiresource Negotiation and Adaptation. *Real-Time Systems*, 15(3):249–273, November 1998.

[12] F. Yu, Q. Zhang, W. Zhu, and Y.-Q. Zhang. QoS-Adaptive Proxy Caching for Multimedia Streaming Over the Internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(3):257–269, March 2003.

[13] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems, EuroSys*, pages 265–278, April 2010.

[14] Y. Zheng, B. Ji, N. Shroff, and P. Sinha. Forget the Deadline: Scheduling Interactive Applications in Data Centers. Technical report, June 2014. http://www2.ece.ohio-state.edu/~zhengy/TR4.pdf.

[15] Y. Zheng, N. Shroff, and P. Sinha. Performance analysis of work-conserving schedulers in minimizing the total flow-time with phase precedence. In *Proceedings of 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton'12)*, October 2012.

[16] Y. Zheng, N. Shroff, and P. Sinha. A new analytical technique for designing provably efficient mapreduce schedulers. In *Proceedings of IEEE INFOCOM*, pages 1600–1608, March 2013.

[17] Y. Zheng, N. Shroff, R. Srikant, and P. Sinha. Exploiting large system dynamics for designing simple data center schedulers. In *Proceedings of IEEE INFOCOM*, April 2015.