Maximizing Online Service Profit for Time-Dependent Applications

Shuhui Li†, Miao Song†, Zheng Li†, Shangping Ren†
Computer Science Department
Illinois Institute of Technology
Chicago, IL 60616
Email:{sli38, msong8, zli80, ren}@iit.edu

Gang Quan‡
Electrical and Computer Engineering Department
Florida International University
Miami, FL 33174
Email:gang.quan@fiu.edu

***Abstract*—As computers and Internet technology advance, many time-dependent applications, such as mobile navigation and online gaming, are emerging. Time-dependent applications are associated with a pair of time-dependent functions representing accrued gain at the time when tasks complete or accrued cost when tasks fail to complete, respectively. For systems that provide time-dependent services, the optimization goal is to maximize the system profit when both potential gain and potential cost exist for each request the system accepts. This paper presents two scheduling algorithms, prediction-based highest gain density first (PHGDF) and iterative PHGDF (iPHGDF) scheduling algorithms with the objective to maximize the system's total accrued profit. Simulation results provide clear evidence that, with respect to the system total accrued profit, the proposed PHGDF and iPHGDF algorithms have advantages over other commonly used scheduling algorithm, such as the Earliest Deadline First (EDF), the Generic Utility Scheduling (GUS) and the Profit and Penalty aware scheduling (PP-aware) algorithms.**

I. Introduction

As computers and Internet technology advance, many time-dependent services are emerging [1], such as mobile navigation by Google and online gaming by Nintendo, to name a few. All of these indicate that a new era of time-dependent on-line services is approaching.

It is worth pointing out that use the term *time-dependent applications* to differentiate from the term *real-time applications*. The main characteristic of real-time applications is that their correctness is determined by both functional correctness and timing correctness. The study of real-time applications often focuses on how to ensure the timing correctness such as ensuring the deadline satisfactions. For time-dependent applications, though they may also have deadline requirements, the deadline itself can be time-dependent. Furthermore, for real-time applications the goal is to meet deadlines and have predictability; there is often no incentive to finish the task early. For time-dependent applications the quality of the applications' end results is often measured by time. The EDF and Rate Monotonic scheduling are the two most important and widely used algorithms [2] for deciding an execution order for a given set of real-time tasks. Jensen et al. proposed to associate each task with a Time Utility Functions (TUF) to indicate the task's time-dependent aspect [3]. Since then much research has been done using accrued value as a scheduling criteria for time-dependent applications.

For online services there is often a service level agreement (SLA) [4] made between service providers and their clients. Service providers are obligated to deliver their service with the agreed level of quality, such as the timeliness for time-dependent applications. Service requesters often base their needed quality level on their request urgency and the cost for increased quality level. The goal of a service provider is to make the most profit possible with the given resources. When the demand for resources exceeds what is available, service providers may not be able to meet all the requests' quality requirements but still be able to make a profit for time-dependent applications. Hence, decisions have to be made as to which requests to honor in order to maximize the service provider's profit.

The task scheduling problem becomes more challenging when a task is selected to execute and its impact on other tasks, from both the cost and gain perspectives, has to be taken into consideration. The Profit and Penalty aware (PP-aware) model and scheduling paradigm are first proposed in [5]. Li et al. [6] extended the work to allow preemption into the schedule and maximize the system's total accrued profit.

In this paper, we study the profit maximization issue for time-dependent applications. In particular, we define a model to abstract time-dependent applications where both gain and possible cost for executing a time-dependent application are taken into consideration. Based on the model, we develop two online scheduling algorithms for maximizing total profit from a given set of applications. We focus on the prediction of non-profit-bearing tasks so that an early action can be taken place to avoid larger a penalty cost. Experimental results show that the two proposed scheduling algorithms can significantly outperform the traditional scheduling approaches.

II. Problem Formulation

A. System Architecture

Assume the online system consists of two modules: the *service request management module* and the *request execution module*.

†The research is supported in part by NSF under grant number CNS-0746643 (CAREER).
‡The research is supported in part by NSF under grant number CNS-0969013, CNS-0917021, and CNS-1018108.

The management module is responsible for (1) deciding the admission of a newly arrived request, (2) selecting the next request for execution, and (3) removing non-profit-bearing requests. The request execution module is not preemptive nor abortable. Both the request management module and the request execution module work concurrently.

B. Time-Dependent Request Model

Use a 4-tuple $(r, e, \mathcal{G}(t), \mathcal{L}(t))$ to define a time-dependent request T, where r and e are the request's release-time and execution time, respectively. The $\mathcal{G}(t)$ and $\mathcal{L}(t)$ are the request's completion time gain and removal time cost functions, respectively, which are defined below. In this paper, the terms task and request are used interchangeably..

Definition 1 (Completion Time Gain Function): The *completion time gain function* $\mathcal{G}(t)$ for request T is a *non-increasing* function of t that denotes the gain obtained when request T is completed at time t. □

Definition 2 (Removal Time Cost Function): The *removal time cost function* $\mathcal{L}(t)$ for request T is a *non-decreasing* function of t that denotes the cost accrued when request T is removed at time t. □

Definition 3 (Non-Profit-Bearing Time Point): Given a request $(r, e, \mathcal{G}(t), \mathcal{L}(t))$, its non-profit-bearing time point D is defined when $\mathcal{G}(t) = 0$, i.e., $D = \mathcal{G}^{-1}(0)$. □

Definition 4 (Profit Function): The profit function for executing request $T = (r, e, \mathcal{G}(t), \mathcal{L}(t))$ is defined as:

$$\mathcal{P}(t) = \begin{cases} 0 & t \le r \\ \mathcal{G}(t) & \text{if } T \text{ is completed at } t, \text{ where } r < t \le D \\ -\mathcal{L}(t) & \text{if } T \text{ is removed at } t, \text{ where } r < t \le D \\ -\infty & \text{if } T \text{ is neither completed} \\ & \text{nor removed after } D \end{cases} \quad (1)$$

□

As defined in (1), if a request is removed at its release time, the service provider makes no profit, or pay any penalty cost. Once the request is accepted for processing, it has the potential to both accrue gain and accrue cost. As time proceeds, gain decreases while cost increases. The overall system performance is evaluated by the total profit obtained when executing a given set of requests.

Definition 5 (Potential Gain Density): Given a request $T = (r, e, \mathcal{G}(t), \mathcal{L}(t))$ and time t_0, its potential gain density at time t_0 is defined as:

$$\eta(t_0) = \frac{\mathcal{G}(t_0 + e)}{e} \quad (2)$$

□

C. Problem Formulation

Problem 1: Given a time-dependent request set $\Gamma = \{T_1, \cdots, T_i, \cdots, T_n\}$, where $T_i = (r_i, e_i, \mathcal{G}_i(t), \mathcal{L}_i(t))$, develop an on-line scheduling algorithm that maximizes a system's total accrued profit, i.e., $\max \sum_{T_i \in \Gamma} \mathcal{P}_i(t_i)$, where t_i is the time at which T_i is completed or removed. □

III. Prediction-based Highest Gain Density First Task Scheduling

A. Admission Test

The purpose of the admission test is to prevent the scenario where a request is accepted but, the system fails to fulfill the SLA causing it to have to pay a penalty for the failure. There are two situations when a new request may be accepted: (1) the system has sufficient resources; or (2) the new request can bring a higher profit than other requests already accepted by the system even after penalty is taken into consideration.

Let $T_N = (r_N, e_N, \mathcal{G}_N(t), \mathcal{L}_N(t))$ denote the newly arrived request, $\mathcal{A}(r_N)$ denote the system's active request set when T_N arrives, and t_0 be the time when requests' gain or cost values are calculated. Algorithm 1 gives the details of the admission process The *for* loop from lines 3 to 12 checks if the potential gain of admitting the new request is greater than the potential cost.

Algorithm 1: ADMISSION-TEST($\mathcal{A}(r_N), T_N, t_0$)

```
1: isAdmit = true
2: sort requests in A(r_N) ∪ {T_N} in descent order based on η_i(t_0), and
   let the sorted order be (T_1, ..., T_{k+1});
3: for i = 1 to k + 1 do
4:     if t_0 + Σ_{j=1}^{i} e_j > D_i then
5:         if T_i ≠ T_N AND
           G_N(t_0 + Σ_{j=1}^{N} e_j) − G_i(t_0 + Σ_{j=1}^{i} e_j) > L_i(r_N) then
6:             remove request T_i;
7:         else
8:             isAdmit = false;
9:             break;
10:        end if
11:    end if
12: end for
13: return isAdmit
```

It is worth pointing out that when T_N arrives, if the execution module is busy executing a request $T_C = (r_C, e_C, \mathcal{G}_C(t), \mathcal{L}_C(t))$, we have to calculate the gain/cost based on the time point when the currently executing request is completed, i.e., $t_0 = t_C + e_C$. Otherwise, $t_0 = r_N$.

B. Prediction-based Highest Gain Density First Task Scheduling

The system's goal is to make maximal profit by executing a set of time-dependent requests. It is not difficult to see that it is a scheduling problem which has proven to be an NP-hard. Hence, for online scheduling, we have to resort to a heuristic approach. The essence behind the *prediction-based highest gain density first* (PHGDF) scheduling algorithm is that at any given scheduling point, it selects and executes the request with the highest unit time potential gain; Based on the selection, it predicts if executing the selected request causes any other request(s) to be non-profit-bearing and if so, removes them.

Let $\mathcal{A}(t_0)$ denote the system active request set at time t_0. Algorithm 2 gives the details for the PHGDF. *find* is the function used to find the first profit request in a sorted request set. The *for* loop from lines 4 to 8 in Algorithm 2 checks if a request will become non-profit-bearing and removes it if it is. It should be noticed that the gain function is defined

within a time range and the profit becomes negative infinity if the completion time of a request is beyond non-profit-bearing point D and needs to be removed (lines 5 to 6).

Algorithm 2: PHGDF$(\mathcal{A}(t_0), t_0)$

1: **sort** requests in $\mathcal{A}(t_0)$ in descent order based on $\eta_i(t_0)$, and let the sorted order be $(T_1, \ldots, T_{|\mathcal{A}(t_0)|})$;
2: T_1 = find$((T_1, \ldots, T_{|\mathcal{A}(t_0)|}))$;
3: dispatch request T_1 to the execution module;
4: **for** i = 2 to $|\mathcal{A}(t_0)|$ **do**
5: **if** $t_0 + \sum_{j=1}^{i} e_j > D_i$ **then**
6: remove request T_i;
7: **end if**
8: **end for**

The complexity of the algorithm comes mainly from the sorting and non-profit-bearing prediction check. Hence, the time complexity of the algorithm is $O(n \lg n)$, where n is the number of requests to be scheduled.

IV. Iterative Prediction-based Highest Gain Density First Task Scheduling Algorithm

The iterative prediction-based highest gain density first (iPHGDF) task scheduling algorithm re-orders the pending queue every time after simulating a request dispatch to make the prediction earlier and hence obtain a better system profit. The admission test and the *find* function for the iPHGDF scheduling algorithm is the same as for the PHGDF approach. Algorithm 3 gives the request selection and removal strategy for the iPHGDF approach.

Algorithm 3: iPHGDF$(\mathcal{A}(t_0), t_0)$

1: **sort** requests in $\mathcal{A}(t_0)$ in descent order based on $\eta_i(t_0)$, and let the sorted order be $(T_1, \ldots, T_{|\mathcal{A}(t_0)|})$;
2: T_1 = find$((T_1, \ldots, T_{|\mathcal{A}(t_0)|}))$;
3: dispatch request T_1 to the execution module;
4: set t be t_0;
5: **while** $\mathcal{A}(t) \neq \emptyset$ **do**
6: **sort** requests in $\mathcal{A}(t)$ in descent order based on $\eta_i(t)$, and let the sorted order be $(T'_1, \ldots, T'_{|\mathcal{A}(t)|})$
7: **for** i = 1 to $|\mathcal{A}(t)|$ **do**
8: **if** $t + \sum_{j=1}^{i} e_j > D_i$ **then**
9: remove request T_i;
10: **end if**
11: **end for**
12: disregard request T'_1 from $\mathcal{A}(t)$;
13: set $t = t + e_1$;
14: **end while**

The time complexity of the iPHGDF is $O(n^2 \lg n)$, where n is the number of requests to be scheduled.

V. Performance Evaluation

In this section, we empirically evaluate the introduced scheduling algorithms, compare them with the optimal solutions obtained with brute-force search for small task sets, and compare them with the Earliest Deadline First (EDF) [2], the Generic Utility Scheduling (GUS) [7], and the Profit and Penalty aware (PP-aware) [5], [6] scheduling approaches for large task sets. The evaluation and comparison focus on two aspects: the system profit and task removal rate.

A. Experiment Setting

The experiments are conducted on a simulator we have developed. In experiments, we assume that both $\mathcal{G}$ and $\mathcal{L}$ are linear functions, and we randomly generate each time-dependent task $T = (r, e, \mathcal{G}(t), \mathcal{L}(t))$ as follows:

- Tasks' release time r is randomly chosen following the Poisson distribution with $\lambda = 5$;
- Tasks' non-profit-bearing time point D is randomly generated, which is uniformly distributed within $[10, 100]$;
- Tasks' execution time e is randomly generated within the range of $[1, u_{max} \times (D - r)]$, where u_{max} is defined below in (3);
- The gradients of $\mathcal{G}$ and $\mathcal{L}$, i.e., a_g and a_l, are randomly generated in the range of $[4, 10]$ and $[1, 5]$, respectively;
- Functions $\mathcal{G}$, $\mathcal{L}$ are defined as follows:

$$\mathcal{G}(t) = \begin{cases} 0 & t < r + e \text{ or } t > D \\ -a_g(t - D) & r + e \leq t \leq D \end{cases}$$

$$\mathcal{L}(t) = \begin{cases} 0 & t < r, \\ a_l(t - r) & r \leq t \leq D \\ \infty & t > D \end{cases}$$

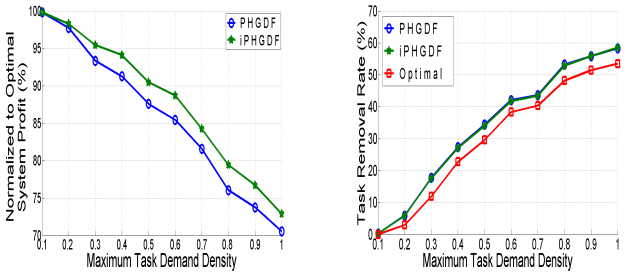
For a given task set $\Gamma = \{T_1, \cdots, T_i, \cdots, T_n\}$, where $T_i = (r_i, e_i, \mathcal{G}_i(t), \mathcal{L}_i(t))$, we use u_{max} to denote the maximum task demand density of the task set, i.e.,

$$u_{max} = \max_{T_i \in \Gamma} \{ \frac{e_i}{D_i - r_i} \} \tag{3}$$

For each of the following experiments, we let u_{max} increase from 0.1 to 1 with step size of 0.1 and generate 100 runs under each u_{max} value. The average values are used in plotting the figures.

B. Performance Comparison with the Optimal Solutions

In this experiment, we randomly generate a task set of size 12. We use brute-force search to find the optimal schedule that results in the maximal system profit and use it as a comparison base. We then apply the PHGDF and iPHGDF algorithms to the same task sets and obtain corresponding system profits and task removal rates. Figure 1(a) shows the system profit normalized to the optimal solutions.



(a) System profit (b) Task removal rate

Fig. 1. Comparison with the optimal solution for small task sets.

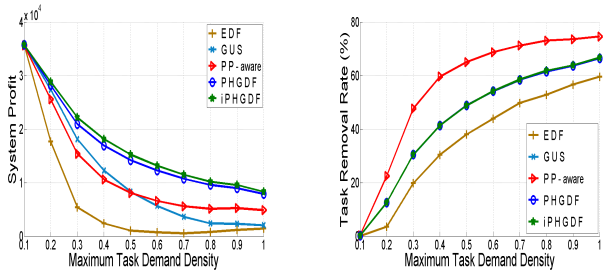
As Figure 1(a) shows, when the u_{max} is low, the profit generated by the algorithms are very close to the optimal solutions obtained through the brute-force search. The reason is that when u_{max} is low most of the requests can finish early and result in a higher system profit. The performances decrease

when u_{max} increases. It is because when time demand of the tasks increases, the competition causes many tasks unable to finish with profit. As our algorithms make choices according to the available system information, though the profits are lower than that of the optimal solutions, PHGDF and iPHGDF can still achieve as much as 71% and 73% of the optimal profit, respectively.

Figure 1(b) shows the average request removal rate compared with the optimal results from the 100 runs for each u_{max}. When u_{max} is low, the system hardly removes tasks. However, as the demand increases, the competition becomes intense, and hence the task removal rate increases even with the optimal solution. When the highest intensity reaches 1, the task removal rate under the PHGDF or iPHGDF approach is about 9.3% higher than the optimal solution's rate.

C. Performance Comparison with EDF, GUS and Profit-Penalty-Aware Approaches

This set of experiments compares the PHGDF and iPHGDF with three existing scheduling approaches: the EDF, GUS, and PP-aware approaches. Similar to previous tests, we randomly generate 100 tasks for each task set. Figure 2(a) and Figure 2(b) depict the system profit and request removal rate under the different algorithms, respectively.



(a) System profit (b) Task removal rate

Fig. 2. Comparison among different algorithms.

From Figure 2(a), it is clear that the PHGDF and iPHGDF scheduling algorithms always outperform the others. It can also be observed from Figure 2(a) that the EDF scheduling algorithm produces the worst performance. This is because EDF gives higher priority to tasks with earlier deadlines without considering the task's potential unit of gain. Hence, even when task demand density is low and all tasks are capable of finishing before their non-profit-bearing point, EDF may cause tasks with high potential gain to finish later than they would have with the other scheduling approaches.

From Figure 2(a), we can observe that the PP-aware algorithm does not have as good of a result as we expected. The reason is that in the PP-aware task model, the task execution time is assumed to be statistically distributed within a given range. The prediction for non-profit-bearing is rather inaccurate as the information itself that the prediction is based upon is statistical. Hence, it can only check for non-profit-bearing requests based on the current selection time and cannot predict anything beyond it. PHGDF and iPHGDF algorithms allow us to look ahead to reduce potential cost or penalty.

Another observation is that when u_{max} is lower than 0.5, the GUS scheduling algorithm has a better performance than the PP-aware scheduling algorithm. When u_{max} is higher than 0.5, the performance ranks change, This is because, the PP-aware scheduling algorithm will choose the highest profit task at scheduling points. The metrics of GUS consider task profit accrued per execution time unit. When u_{max} is low and most tasks can be finished, considering the rate of obtaining profit is more useful. Hence the GUS has better results than the PP-aware scheduling algorithm. When u_{max} is high and more tasks will be removed, considering the amount of obtained profit is a better choice. In this case, the PP-aware has better results than GUS.

For the request removal rate, because EDF gives higher priority to tasks with earlier deadlines, it has the lowest request removal rate among all the scheduling algorithms. PHGDF scheduling, iPHGDF scheduling, and GUS scheduling algorithms have the same request removal rate. They all make decisions based on the active task potential gain density. Only the removal times are different, which result in different system profits. However, as the PP-aware scheduling algorithm only considers profit at the scheduling points, it has the highest request removal rate.

VI. Conclusion

In this paper, we have presented two online scheduling algorithms for time-dependent applications, i.e., PHGDF and iPHGDF scheduling algorithms. The objective of these algorithms is to maximize a system's total accrued profit. Our simulation results have shown that both PHGDF and iPHGDF algorithms can accrue more profit than the common scheduling algorithms. The iPHGDF scheduling algorithm performs slightly better than the PHGDF algorithm. The time complexity of the iPHGDF algorithm is also slightly higher than that of the PHGDF algorithm. They are at $O(n \lg n)$ and $O(n^2 \lg n)$, respectively.

References

[1] F. Casati and M. Shan, "Definition, execution, analysis and optimization of composite e-service," *IEEE Data Engineering*, 2001.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[3] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *Proceedings of the 6th IEEE Real-time systems symposium*, 1985, pp. 112–122.

[4] C. S. Yeo and R. Buyya, "Service level agreement based allocation of cluster resources: Handling penalty to enhance utility," in *Cluster Computing, 2005. IEEE International*, 2005, pp. 1–10.

[5] Y. Yu, S. Ren, N. Chen, and X. Wang, "Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 334–339.

[6] S. Li, S. Ren, Y. Yu, X. Wang, L. Wang, and G. Quan, "Profit and penalty aware scheduling for real-time online services," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 1, pp. 78–89, Feb. 2012.

[7] P. Li, H. Wu, B. Ravindran, and E. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *Computers, IEEE Transactions on*, vol. 55, no. 4, pp. 454–469, April 2006.