Information Processing Letters 46 (1993) 129–133
Elsevier

11 June 1993

An algorithm for coalescing operations with precedence constraints in real-time systems

Lung-Tien Liu

Telecommunication Laboratories, P.O. Box 71, Chung-Li, Taiwan, ROC

Gen-Huey Chen

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC

Kwei-Jay Lin

Department of Computer Science, University of Illinois, Urbana, IL, USA

Communicated by D. Gries
Received 14 November 1991
Revised 20 July 1992 and 26 February 1993

Keywords: Algorithms; dynamic programming; maximum weighted compatible matching; real-time systems

1. Introduction

Real-time computations have deadline constraints. In addition to providing a result with a correct value, a real-time computation must produce the result before the deadline. A real-time system usually has many jobs sharing system resources, such as CPU and I/O devices. The system must provide a feasible schedule for all jobs so that they can finish executions before their deadlines. In many applications [1,3], real-time systems are modeled as object-oriented systems. In an object-oriented system, each object has a set of well-defined operations. Each object also has local variables, which may be accessed only by the operations defined in the interface of the object. The object decides if and when to process the requests from other objects.

In real-time systems, an object usually makes scheduling decisions in order to maximize the system performance. There are many ways to make scheduling decisions. *Operation coalescence* is one of them [2]. The idea is that some objects may be able to handle several distinct requests at the same time. For example, suppose a stack object is implemented in a system and suppose the public, or primitive, operations defined for the stack object are: *push, pop, new* and *top*. We may provide a coalesced operation *double-push*, which takes two elements and pushes both of them onto the stack at the same time. Whenever two consecutive *push* operations are found in front of the request queue for the stack object, the object scheduler can invoke the coalesced

Correspondence to: Professor G.-H. Chen, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC.

operation *double-push* instead. For many distributed systems using the client-server model, operation coalescence can be easily adopted.

By coalescing several requests into one single request, the workload of the object may be reduced. Moreover, the response time for a job waiting for the result from the object may be shortened if its request is coalesced with an earlier request from another job. It is true that other requests in the middle may be delayed if a later request is coalesced with an earlier request. It is therefore the responsibility of the object or the system scheduler to decide if a coalescence should be performed. This is similar to the proposal of using non-conventional protocols to handle real-time transactions [4]. By executing urgent transactions first [5] and delaying non-urgent requests, a real-time system can provide better performance by meeting more deadlines.

In most applications, coalesced operations must be defined by the programmer, due to the semantic issues involved. On the other hand, whether a coalescence should be performed is usually decided by the object (or system) scheduler. In the scheduler, the *reward* (usually the amount of computation time saved) for each coalesced operation must be pre-analyzed and recorded in a *reward table* before execution. When a sequence of requests arrive, the scheduler consults the reward table and determines which requests to coalesce in order to maximize the total reward.

In a real-time system, the request patterns of many jobs are known in advance, especially when jobs are periodic. For example, service requests from a radar monitor are periodic and well-defined. Given K periodic jobs and the reward table, we can determine which requests to coalesce to produce the maximum total reward. In other words, we can find the coalescence schedule that saves the most time. If some requests are not predefined, we can still perform the analysis at run-time if the complexity of scheduling coalesced operations is not high. Unfortunately, finding an optimal schedule in the general case is NP-hard [1]. To simplify the problem, we assume that only two primitive operations can be coalesced at a time.

In this paper, we consider the model in which each periodic job J_i has n sequential operations $\{J_{i,1}, J_{i,2}, \ldots, J_{i,n}\}$ and execution of each job must satisfy the precedence constraint, i.e., each operation $J_{i,j}$ cannot be executed until operation $J_{i,j-1}$ has completed, $1 < j \leqslant n$. For any two operations, there is a reward value. If the two operations cannot be coalesced, the reward value is zero. Our objective is to find a feasible coalescence schedule that produces the maximum total reward.

For the case of two periodic jobs, Chen et al. [1] have developed an $O(n^6)$ time algorithm for the problem. Using dynamic programming, we improve their work by presenting an $O(n^2)$ time algorithm. Moreover, the algorithm can be extended to handle any k periodic jobs with time complexity $O(k^2 n^k)$.

2. An $O(n^2)$ time algorithm for scheduling two periodic jobs

We assume that there are two periodic jobs in the real-time system, and each job J_i has n sequential operations $\{J_{i,1}, J_{i,2}, \ldots, J_{i,n}\}$, $i = 1, 2$. To describe the problem formally, we represent the system as an undirected graph. Each vertex $v_{i,j}$ denotes an operation $J_{i,j}$. There are edges between $v_{i,j}$ and $v_{i,j+1}$, $i = 1, 2$, $j = 1, \ldots, n-1$, and between $v_{1,p}$ and $v_{2,q}$, $p = 1, \ldots, n$, $q = 1, \ldots, n$. Each edge is associated with a weight representing the reward value. Thus, maximizing the total reward is equivalent to finding a *maximum weighted compatible matching* [1] in the graph. Any two edges $(v_{1,r}, v_{2,s})$, $(v_{1,p}, v_{2,q})$ in a weighted compatible matching satisfy either $r < p$ and $s < q$ or $r > p$ and $s > q$. In other words, the maximum weighted compatible matching problem is to find a subset of edges that do not cross each other and whose total weight is maximized. This subset of edges is called the *maximum weighted compatible matching set*.

As an example, Fig. 1 shows a real-time object with four primitive operations op_1, op_2, op_3, and op_4. There are two periodic jobs and each has four operations. Job J_1 consists of op_1, op_4, op_2, and op_3 ($J_{1,1}$, $J_{1,2}$, $J_{1,3}$, and $J_{1,4}$, respectively).

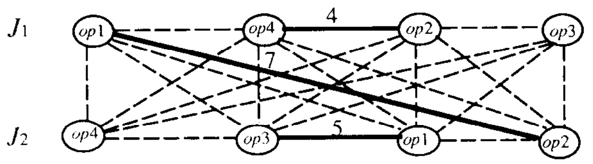

Fig. 1. An example of scheduling operations with precedence constraints.

Job J_2 consists of op_4, op_3, op_1, and op_2 ($J_{2,1}$, $J_{2,2}$, $J_{2,3}$ and $J_{2,4}$, respectively). The reward values are given in Table 1. The maximum weighted compatible matching set is indicated as solid lines in Fig. 1. The maximum total reward is 16 and the scheduling sequence is $J_{2,1}$, $J_{2,2}+J_{2,3}$, $J_{1,1}+J_{2,4}$, $J_{1,2}+J_{1,3}$, $J_{1,4}$.

Let $R_{i,j}$ denote the maximum total reward for matching operations in $\{J_{1,1},\dots,J_{1,i}\}$ and $\{J_{2,1},\dots,J_{2,j}\}$, $r_{i,j}$ the weight of edge $(v_{1,i}, v_{2j})$, $1 \leqslant i \leqslant n$, $i \leqslant j \leqslant n$, and $c_{k,l}$ the weight of edge $(v_{k,l-1}, v_{k,l})$, $k=1, 2$, $1<l \leqslant n$. Clearly, considering operations $J_{1,i}$ and $J_{2,j}$, we have the following equations:

$$R_{i,j}=\max\{R_{i-1,j}, R_{i,j-1}, c_{1,i}+R_{i-2,j}, c_{2,j}+R_{i,j-2}, r_{i,j}+R_{i-1,j-1}\} \quad \text{for } 2 \leqslant i \leqslant n,\ 2 \leqslant j \leqslant n, \tag{1}$$

$$R_{1,j}=\max\{R_{0,j}, R_{1,j-1}, c_{2,j}+R_{1,j-2}, r_{1,j}+R_{0,j-1}\} \quad \text{for } 2 \leqslant j \leqslant n, \tag{2}$$

$$R_{i,1}=\max\{R_{i-1,1}, R_{i,0}, c_{1,i}+R_{i-2,1}, r_{i,1}+R_{i-1,0}\} \quad \text{for } 2 \leqslant i \leqslant n, \tag{3}$$

$$R_{0,j}=\max\{R_{0,j-1}, c_{2,j}+R_{0,j-2}\} \quad \text{for } 2 \leqslant j \leqslant n, \tag{4}$$

Table 1
An example of the reward table

	$op1$	$op2$	$op3$	$op4$
$op1$	2	7	5	0
$op2$	7	2	0	4
$op3$	5	0	2	1
$op4$	0	4	1	2

$$R_{i,0}=\max\{R_{i-1,0}, c_{1,i}+R_{i-2,0}\} \quad \text{for } 2 \leqslant i \leqslant n, \tag{5}$$

$$R_{0,0}=R_{1,0}=R_{0,1}=0, \quad R_{1,1}=r_{1,1} \text{ if } r_{1,1}>0, \text{ and } R_{1,1}=0 \text{ otherwise.} \tag{6}$$

In (1), the first (second) term in the right-hand side represents the case that operation $J_{1,i}$ ($J_{2,j}$) is not coalesced with any other operation. The third (fourth) term represents the case that operation $J_{1,i}$ ($J_{2,j}$) is coalesced with its immediate predecessor. The fifth term represents the case that $J_{1,i}$ is coalesced with $J_{2,j}$. No other cases are possible, because edges in the maximum weighted compatible matching set are not allowed to cross each other due to precedence constraints. The other equations are derived similarly.

With initial values given by (6), the following algorithm takes $O(n^2)$ time to compute the value $R_{n,n}$, which is the maximum total reward.

```
/* Initialization */
$R_{0,0} := 0$; $R_{1,0} := 0$; $R_{0,1} := 0$;
**if** $r_{1,1} > 0$ **then** $R_{1,1} := r_{1,1}$ **else** $R_{1,1} := 0$;
**for** $i := 2$ **to** $n$ **do**
    $R_{i,0} := \max\{R_{i-1,0}, c_{1,i}+R_{i-2,0}\}$;
**for** $j := 2$ **to** $n$ **do**
    $R_{0,j} := \max\{R_{0,j-1}, c_{2,j}+R_{0,j-2}\}$;
**for** $i := 2$ **to** $n$ **do**
    $R_{i,1} := \max\{R_{i-1,1}, R_{i,0}, c_{1,i}+R_{i-2,1},$
            $r_{i,1}+R_{i-1,0}\}$;
**for** $j := 2$ **to** $n$ **do**
    $R_{1,j} := \max\{R_{0,j}, R_{1,j-1}, c_{2,j}+R_{1,j-2},$
            $r_{1,j}+R_{0,j-1}\}$;

/* Compute $R_{n,n}$ */
**for** $i := 2$ **to** $n$ **do**
    **for** $j := 2$ **to** $n$ **do**
        $R_{i,j} := \max\{R_{i-1,j}, R_{i,j-1}, c_{1,i}+R_{i-2,j},$
                $c_{2,j}+R_{i,j-2}, r_{i,j}+R_{i-1,j-1}\}$.
```

3. Extension to k periodic jobs

We now extend our result to $k \geqslant 3$ periodic jobs. We assume that each job J_i, $1 \leqslant i \leqslant k$, has n

sequential operations $\{J_{i,1}, J_{i,2}, \ldots, J_{i,n}\}$. Let $R_{t[1],t[2],\ldots,t[k]}$ denote the maximum total reward for matching operations in $\{J_{1,1}, \ldots, J_{1,t[1]}\}$, $\{J_{2,1}, \ldots, J_{2,t[2]}\}, \ldots, \{J_{k,1}, \ldots, J_{k,t[k]}\}$, $r_{i,j,p,q}$ the weight of edge $(v_{i,j}, v_{p,q})$, $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant n$, $1 \leqslant p \leqslant k$, $1 \leqslant q \leqslant n$, $i \neq p$, and $c_{u,v}$ the weight of edge $(v_{u,v-1}, v_{u,v})$, $1 \leqslant u \leqslant k$, $1 < v \leqslant n$.

Depending on whether operations in $S = \{J_{1,t[1]}, J_{2,t[2]}, \ldots, J_{k,t[k]}\}$ are coalesced with some other operations, there are three possible cases.

Case 1: Some operation in S is not coalesced with any other operation. We have

$$\begin{aligned} M_1 &= R_{t[1],t[2],\ldots,t[k]} \\ &= \max\{R_{t[1]-1,t[2],\ldots,t[k]}, R_{t[1],t[2]-1,\ldots,t[k]}, \\ &\qquad \ldots, R_{t[1],t[2],\ldots,t[k]-1}\}. \end{aligned}$$

Case 2: Some operation in S is coalesced with its immediate predecessor. We have

$$\begin{aligned} M_2 &= R_{t[1],t[2],\ldots,t[k]} \\ &= \max\{c_{1,t[1]} + R_{t[1]-2,t[2],\ldots,t[k]}, \\ &\qquad c_{2,t[2]} + R_{t[1],t[2]-2,\ldots,t[k]}, \ldots, \\ &\qquad c_{k,t[k]} + R_{t[1],t[2],\ldots,t[k]-2}\}. \end{aligned}$$

Case 3: Two operations in S are coalesced. If $J_{1,t[1]}$ is coalesced with $J_{2,t[2]}$, the maximum total reward is equal to $r_{1,t[1],2,t[2]} + R_{t[1]-1,t[2]-1,\ldots,t[k]}$. Thus, we have

$$\begin{aligned} M_3 &= R_{t[1],t[2],\ldots,t[k]} \\ &= \max\{r_{1,t[1],2,t[2]} + R_{t[1]-1,t[2]-1,\ldots,t[k]}, \\ &\qquad r_{1,t[1],3,t[3]} + R_{t[1]-1,t[2],t[3]-1,\ldots,t[k]}, \ldots, \\ &\qquad r_{1,t[1],k,t[k]} + R_{t[1]-1,t[2],\ldots,t[k]-1}, \\ &\qquad r_{2,t[2],3,t[3]} + R_{t[1],t[2]-1,t[3]-1,\ldots,t[k]}, \\ &\qquad r_{2,t[2],4,t[4]} + R_{t[1],t[2]-1,t[3],t[4]-1,\ldots,t[k]}, \\ &\qquad \ldots, r_{2,t[2],k,t[k]} + R_{t[1],t[2]-1,t[3],\ldots,t[k]-1}, \\ &\qquad \ldots, r_{k-1,t[k-1],k,t[k]} \\ &\qquad\quad + R_{t[1],t[2],\ldots,t[k-1]-1,t[k]-1}\}. \end{aligned}$$

Due to precedence constraints, no other cases are possible. Therefore, $R_{t[1],t[2],\ldots,t[k]} = \max\{M_1, M_2, M_3\}$. For example, given operations $\{J_{1,1}, \ldots, J_{1,4}\}$, $\{J_{2,1}, \ldots, J_{2,4}\}$, and $\{J_{3,1}, \ldots, J_{3,4}\}$,

$$M_1 = \max\{R_{3,4,4}, R_{4,3,4}, R_{4,4,3}\},$$

$$M_2 = \max\{c_{1,4} + R_{2,4,4}, c_{2,4} + R_{4,2,4}, c_{3,4} + R_{4,4,2}\},$$

and

$$\begin{aligned} M_3 = \max\{&r_{1,4,2,4} + R_{3,3,4}, r_{1,4,3,4} + R_{3,4,3}, \\ &r_{2,4,3,4} + R_{4,3,3}\}. \end{aligned}$$

The maximum total reward is

$$\begin{aligned} R_{4,4,4} &= \max\{M_1, M_2, M_3\} \\ &= \max\{R_{3,4,4}, R_{4,3,4}, R_{4,4,3}, \\ &\qquad c_{1,4} + R_{2,4,4}, c_{2,4} + R_{4,2,4}, \\ &\qquad c_{3,4} + R_{4,4,2}, r_{1,4,2,4} + R_{3,3,4}, \\ &\qquad r_{1,4,3,4} + R_{3,4,3}, r_{2,4,3,4} + R_{4,3,3}\}. \end{aligned}$$

It is not difficult to show that the time complexity of computing the maximum total reward $R_{n,\ldots,n}$ is $O(k^2 n^k)$.

4. Discussion and conclusion

Using dynamic programming, we have developed an $O(n^2)$ time algorithm for the proposed problem. The same technique can be easily applied to the case of coalescing three or more operations. It is not difficult to see that $O(n^3)$ time is sufficient for the case of coalescing three operations.

Although we made a significant improvement over Chen et al.'s work, we have not determined what the lower bound for the problem is.

References

[1] T.E. Bihari and P. Gopinath, Object-oriented real-time systems: Concept and examples, *IEEE Comput.* **25** (12) (1992) 25-32.

[2] M.I. Chen, J.Y. Chung and K.J. Lin, Scheduling algorithm for coalesced operations in real-time systems, in: *Proc. COMPSAC 89*, Orlando, FL (1989) 143-150.

[3] K.B. Kenny and K.J. Lin, Structuring large real-time systems with performance polymorphism, in: *Proc. IEEE Real-Time Systems Symp.* (1990) 238–246.

[4] T.W. Kuo and A.K. Mok, Application semantics and concurrency control of real-time data-intensive applications, in: *Proc. IEEE Real-Time Systems Symp.*, Phoenix, AZ (1992) 35–45.

[5] C.L. Liu and J.W. Layland, Scheduling algorithm for multiprogramming in a hard real-time environment, *J. ACM* **20** (1) (1973) 46–61.