Hard/Soft Deadline Assignment for High Workflow Throughput

Jin Hyun Son, Jung Hoon Kim, and Myoung Ho Kim
Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1, Kusung-Dong, Yusung-Gu, Taejon, 305-701, Korea
{jhson, kimjh, mhkim}@dbserver.kaist.ac.kr

Abstract

Many business processes which are abstracted to workflows have time constraints such that their processing should be completed within their deadlines. New time management technologies are, therefore, required for high workflow throughput which can maximize the number of workflow instances satisfying their deadlines. In this paper we propose an efficient deadline assignment method and validate its usefulness with experiments. Especially, the method divides activity deadlines into two categories, namely hard and soft deadlines according to the characteristics of activities. This can improve workflow throughput more and more.

1. Introduction

Nowadays, most business processes can be simplified, computerized, and automated by the help of a workflow. The improvement of workflow relevant technologies such as middleware and object-oriented techniques makes workflow management systems available and applicable to many real applications. A workflow consists of a set of activities interconnected by workflow control structures: sequence, AND, OR, and LOOP. Hence, there can be many execution paths in a workflow in terms of its context. Each activity has an agent which performs the role of the activity. An activity is called a human interactive activity or a system automatic activity according as its agent is human or a software process, respectively.

In practice, many business processes have time constraints and some works on time-constrained workflows have been done [2][6]. During workflow executions, if a workflow instance cannot meet the activity deadline, exceptional operations called escalation may be made for the instance. In general, escalation includes compensations of already completed activities which may waste shared system resources, give additional costs to workflow management system, and finally degrade workflow throughput.

For the purpose of high workflow throughput, we should minimize the number of escalated workflow instances, and might as well escalate workflow instances as early as possible if escalation seems to be unavoidable. Early escalation has been studied in the predictive workflow [5]. In this paper we are concerned with minimizing the number of escalated workflow instances by effectively allocating activity deadlines. Activity deadlines can be determined statically or dynamically. Static activity deadlines are determined in a workflow definition, while dynamic deadline assignment methods may adjust the static deadlines according to workflow system environments such as system workloads during workflow executions [6][5]. Basically, the dynamic methods proposed in [6] and [5] distribute the slack time obtained at an activity, which is the difference between its static deadline and actual execution time, among its subsequent activities. Even if these methods can determine activity deadlines by reflecting dynamic workflow system environments, they need so much deadline computing time for each activity as well as every workflow instance. After all, workflow throughput may be degraded due to the excessive computing time. Hence, only if we can estimate and determine static activity deadlines at the workflow build-time while precisely predicting system workloads, we can maximize the number of workflow instances completed timely.

In this objective, we first categorize activities into critical and non-critical activities by the concept of the critical path which is the longest execution path in a workflow. Critical activities may have hard deadlines while non-critical activities have soft deadlines. We expect that critical activities will include most bottleneck points in the workflow. If a workflow instance cannot meet the hard deadline of a critical activity, we escalate the workflow instance since the possibility not to meet the workflow deadline is very high. On the other hand, if a workflow instance cannot meet the soft deadline of a non-critical activity, we had better make it continue to perform its works. This can somewhat reduce the possibility that some workflow instances which will be

0-7695-0496-5/00 $10.00 © 2000 IEEE

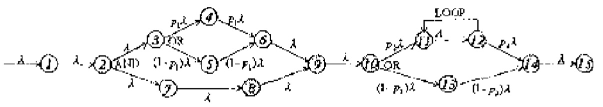


Figure 1. Workflow Queueing Network

completed within the workflow deadline may be escalated due to violating the activity deadline in any intermediate activity. We can, therefore, apply different escalation policies according to the activity group. Next, we propose an effective static deadline assignment method called Proportional to the Sojourn Time (PST) which is based on the execution time of each activity, i.e., service time plus waiting time. Finally, we show the validness of our scheme for managing activity deadlines with experiments.

Workflow management systems allow to define the deadlines of a workflow and its activities in a workflow definition [4]. Nowadays, many business and scientific workflow applications have adopted time management technologies. Originally, the deadline assignment problem has been intensively studied in distributed soft real-time systems in which a task is composed of subtasks [3]. [3] introduced four deadline assignment methods: Ultimate Deadline (UD), Effective Deadline (ED), Equal Slack (EQS), and Equal Flexibility (EQF). Under UD, a task and all its subtasks have the same deadlines. In ED, the deadline of any subtask is the time that is the deadline of the task minus the total expected execution time of its subsequent subtasks. These two methods determine activity deadlines statically. EQS divides the total remaining slack equally among the remaining subtasks. And, EQF divides the total remaining slack among the subtasks in proportion to their execution times. These two methods can be applied to both static and dynamic deadline assignment. Because a task is to subtasks in distributed soft real-time systems what a workflow is to activities in workflows, four deadline assignment methods above can be easily applied to workflow systems [6][5]. On the other hand, [6] proposed four dynamic deadline assignment methods based on the static activity deadlines: Total Slack (TSL), Under Proportional Execution (PEX), Proportional Escalation (PES), and Proportional Load (PLO).

In section 2, we describe the workflow model considered in this paper. Section 3 explains how to find out critical and non-critical activities. Section 4 presents our static deadline assignment method, and section 5 analyzes the method with several experimental results. Finally, we conclude our paper with its summary and further works in section 6.

2. Workflow Model

A workflow is a network consisting of activities interconnected with each other as in Figure 1. Each activity has an agent to perform its role. In this paper we assume that the arrival process of workflow service requests is a Poisson process and the agent of each activity has an exponential service time. This can make a workflow modeled as a M/M/1 queueing network in which each activity is an independent M/M/1 queueing system. We can specify the arrival and departure process in each activity as in Figure 1 with the information of the initial arrival rate to the workflow, the service rate of each activity agent, and the branch probability in OR and LOOP. The decomposition and superposition of independent Poisson processes in the OR control structure, e.g., activity 10 and 14 of Figure 1 are known to be also Poisson processes from time reversibility [8]. Even though the actual internal flow of a LOOP is not a Poisson process due to its feedback, it is known that this structure behaves as if its activities are independent M/M/1's [1]. In the AND control structure, the departure processes of an AND split activity, e.g., activity 2 are clearly Poisson processes, but the arrival process of an AND join activity, e.g., activity 9 is not actually a Poisson process because all requests coming into the activity must be synchronized before its agent is invoked. Because a point of synchronized time is greatly determined by the path with the longest average execution time of n paths in the AND, the arrival process of an AND join activity can be approximated by the Poisson process of the requests going through the longest execution path. Hence, a workflow can be modeled by a M/M/1 queueing network with Poisson arrival and departure processes.

It is difficult to understand workflow schemas and find their faults if they are designed by specifying arbitrary relationships between activities in a workflow definition [6]. In this regard, we define a nested workflow with structural forms and only consider workflow schemas defined with it.

Definition 2.1 *When a non-sequential control structure in a workflow contains another workflow control structures within it, the workflow is called the* ***nested workflow****. Here, when a LOOP is contained within some non-sequential control structure C, the destination of the feedback must be somewhere within C.*

Definition 2.2 *The outermost non-sequential control structure in a nested workflow is called a non-sequential control block, or simply* ***control block****.*

There are two control blocks in Figure 1, an AND control block containing an OR and an OR control block with a LOOP. If the destination of the feedback of the LOOP is an activity 6 instead of activity 11, the workflow does not have meaningful control flows. Thus, we only consider a nested workflow in this paper.

In the following, we distinguish between critical and non-critical activities in a workflow. And then, our static deadline assignment method called PST determines the hard or soft deadline of each activity. We can apply different escalation policies to workflow management systems according to the hard or soft deadline.

3. Critical/Non-critical Activities

As a workflow management system creates a new workflow instance for an user's workflow request, there can be many workflow instances which are concurrently executed and some workflow instances may wait at the queue of an activity during other workflow instances to arrive ahead are served. This means that *the average execution time* of a workflow instance in an activity is *the average service time* of the agent for the activity plus *the average waiting time* of the workflow instance at the activity.

How to distinguish between critical or non-critical activities is dependent on workflow designers' decisions. For example, a workflow designer can select some critical activities by heuristic and analytical methods based on the workflow schema. In this paper we use the concept of the critical path for the classification. The critical path is a sequence of activities from the beginning to the end of a workflow such that has the longest average execution time. If an activity belongs to the critical path, it is called a critical activity. Otherwise, it is a non-critical activity. Our previous work [7] proposed a method to find out the critical path which is based on the characteristics of workflow control structures as well as the average execution time of a workflow instance in an activity. The method is called the Innermost Control Structure First (ICSF) and its algorithm is described in Figure 2.

ICSF first determines a sub-critical path with the longest average execution time in each control block, and then combines these sub-critical paths. When we determine a sub-critical path in each control block, the longest execution path is selected from the innermost control structure to the outermost control structure. If the innermost control structure is an AND or OR, we select a path with the longest average execution time. If a LOOP control structure, we transform it into a sequential control structure which we can analyze with ease. All sequential control structures except ones within control blocks belong to the critical path because the sequences are all unique paths in the workflow. Because the average execution time is known as $W = \frac{1}{\mu - \lambda}$ in M/M/1 with the arrival rate λ and the service rate μ, the longest execution path in an AND or OR control structure as in Figure 3 is determined as follows. With $W_i = \frac{1}{\mu_i - \lambda_i}$ of the average execution time in an activity i, the longest execution path is a sequential path with $MAX(\sum \frac{1}{\mu_i - \lambda_i})$ for all activities within each path, where the arrival rate λ_i and the service rate μ in activity i. Transforming a LOOP into a sequence will be presented in section 4.

As an example to find out the critical path using the

```
1. All sequential control structures except ones within **control
   blocks** belong to the critical path.

2. WHILE (each control block)
   {
     3. IF (the innermost control structure in the control block
             is an AND or OR)
             Select a path with the longest average execution
             time in the control structure.
     4. IF (the innermost control structure in the control block
             is a LOOP)
             Transform it into a sequential control structure.
     5. IF (a sub-critical path of the control block is not yet
             determined)
             GO TO 3.
   }

6. The critical path is obtained by combining all the subcritical
   paths.
```

Figure 2. ICSF Algorithm

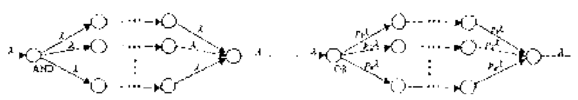


Figure 3. AND or OR Control Structure

ICSF, there are two control blocks in Figure 1: an AND and OR control block. Innermost control structures are OR and LOOP, respectively. We first determine a sub-critical path in the OR control structure and transform the LOOP into a sequence. After determining sub-critical paths for two control blocks, we can find out the critical path by combining all the sub-critical paths.

4. Deadline Assignment Method

Under the high arrival rate for requesting workflow services, the time that a workflow instance waits for the service at an activity is usually much more longer than the time that the instance is served at the activity. If we allocate activity deadlines without considering the waiting time, many workflow instances will not meet their deadlines to be finally escalated, which may well degrade workflow throughput. In this respect, we propose a new deadline assignment method called Proportional to the Sojourn Time (PST) using the average execution time in an activity and considering the characteristics of workflow control structures. Because an activity basically needs its service time, PST determines the static deadline of an activity by means of giving the activity some additional slack time adding to its service time.

Definition 4.1 *Workflow slack time = Workflow deadline - Longest average execution time among workflow paths*

Eventually, our PST divide the workflow slack time among activities in proportion to their average execution times, which means that the workflow slack time is equal to the sum of all activity slack times. In the following, we explain the PST method with in-depth analyses for workflow control structures.

```
STEP1: Transform a nested workflow into a sequential workflow using
       the execution time of an activity
       WHILE (for all control blocks) {
          1. IF (the innermost control structure is a LOOP)
                - Transform the LOOP into a sequence
                - Substitute an activity for the sequence
          2. IF (the innermost control structure is an AND or OR)
                - In an AND or OR with n paths, transform each path
                  into an activity
                - Transform n paths into an activity
          3. IF (there exist any control blocks not to be transformed into
                an activity)
                GO TO 1
       }
       /* We obtain a new sequential workflow. */

STEP2: Reallocate activity slack times
       4. Allocate the workflow slack time to activities of the sequential
          workflow in proportion to their execution times
       WHILE (for all activities corresponding to control blocks in the original
                nested workflow) {
          5. IF (the outermost control structure to which the activity slack time
                is not reassigned yet is a LOOP in a control block)
               - Divide the slack time among activities, which belong to
                 the sequence transformed from a LOOP in STEP1,
                 in proportion to their execution times
          6. IF (the outermost control structure to which the activity slack time
                is not reassigned yet is an AND or OR
                in a control block)
               - Equally allocate the slack time as the total slack time of each
                 path in an AND or OR
               - In each path with a sequence, divide its slack time among
                 activities in proportion to their execution times
          7. IF (there exist control structures to which the slack time is not
                reassigned)
               GO TO 6
       }
```

Figure 4. PST Algorithm

The PST algorithm is composed of two steps as in Figure 4. STEP1 transforms a nested workflow into a workflow with only a sequential control structure. With this sequential workflow, we divide the workflow slack time among activities in proportion to their execution times in STEP2. In this process, slack times of activities corresponding to control blocks or workflow control structures are reallocated to activities within them in the reverse order of the transformation. Figure 5 shows the result of STEP1 which is the sequential workflow transformed from the nested workflow depicted in Figure 1. AND and OR control blocks are replaced with activity $T1$ and $T2$, respectively, by the WHILE of STEP1. Figure 6 explains the process of transforming the AND control block into activity $T1$ and reallocating the activity slack time to activities within the control block in the reverse order of the transformation. During STEP1, activity 4 and 5 of the innermost OR control structure are first transformed into activity $t1$. And then, two sequential control structures are transformed into activity $t2$ and $t3$, respectively. Finally, we can obtain activity $T1$ corresponding to the control block by transforming activity $t2$ and $t3$ into $T1$. The OR control block can be also transformed into activity $T2$ in the same process as the AND control block.

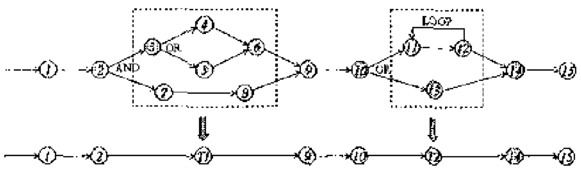
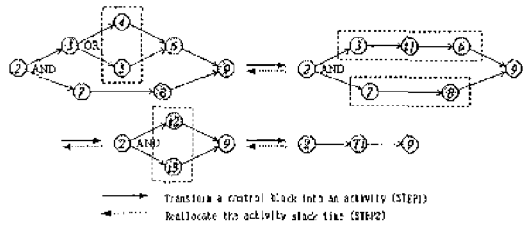


Figure 5. Workflow transformation



AND
OR
Transform a control block into an activity (STEP1)
Reallocate the activity slack time (STEP2)

Figure 6. Transformation and reassignment

STEP2 starts with the sequential workflow from STEP1. First, we distribute the workflow slack time among activities which belong to the sequential workflow in proportion to their execution times. For each activity corresponding to a control block, e.g., activity $T1$ and $T2$ in Figure 5, we reallocate the activity slack time to activities included in the control block. After all, we can distribute the workflow slack time among all activities within a nested workflow and determine their static deadlines. Next, we demonstrate how to transform sequence, AND, and OR control structures into an activity and a LOOP into a sequence. These are essential for the PST algorithm, as shown above.

Sequential Control Structure:

In STEP1, a sequential control structure composed of n activities is transformed into activity t as in Figure 7. For the transformation, we must determine the service rate μ' of activity t satisfying that the average execution time of activity t is equal to that of the sequence. Because the average execution time is $W = \frac{1}{\mu - \lambda}$ in a M/M/1 with the arrival rate λ and service rate μ, the average execution time of a sequence is the sum of the average execution times of all activities included in the sequence [8]. Hence, when the average execution time of activity i is $W_i = \frac{1}{\mu_i - \lambda}$ ($i = 1, ..., n$), the service rate μ' of activity t can be obtained as: $\mu' = \frac{1}{\sum_{i=1}^{n} W_i} + \lambda$ from $\sum_{i=1}^{n} W_i = \frac{1}{\mu' - \lambda}$. On the other

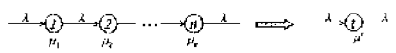


Figure 7. Transformation of a sequence

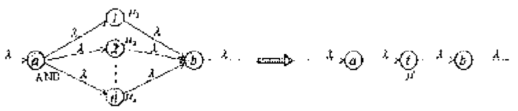


Figure 8. Transformation of an AND control structure

hand, we divide the slack time allocated to activity t among n activities of the sequence in proportion to their average execution times.

AND Control Structure:

Because each sequence path in an AND control structure can be transformed into an activity, we just consider an AND with a single activity per path as in Figure 8. In STEP1, an AND control structure is transformed into an activity which has the same average execution time as the AND. Since n sub-requests of the AND must be synchronized at the AND join activity, i.e., activity b, the average execution time of the AND is dependent on the longest execution path. Let random variables on the average execution times of activity $1, 2, ..., n$ in Figure 8 be $X_1, X_2, ..., X_n$, respectively. With the average execution time $W_i = \frac{1}{\mu_i - \lambda}$ for activity i, the probability density function $f_{X_i}(t)$ of X_i is $f_{X_i}(t) = 1 - e^{-(\mu_i - \lambda)t}$, $i = 1, ...n$. When Y is a random variable on the average execution time of the AND, $Y = max(X_1, X_2, ..., X_n)$ and the distribution function $G_Y(x)$ is

$$\begin{aligned} G_Y(x) &= P(Y < x) \\ &= P(max(X_1, X_2, ..., X_n) < x) \\ &= P(X_1 < x, X_2 < x, ..., X_n < x) \\ &= (1 - e^{-\frac{x}{W_1}})(1 - e^{-\frac{x}{W_2}})...(1 - e^{-\frac{x}{W_n}}) \end{aligned}$$

After all, we can estimate the service rate μ' of activity t which is corresponding to the AND control structure in Figure 8 using the average execution time of the AND, $E[Y] = \int_0^\infty 1 - G_Y(x)\,dx$. Because of $E[Y] = \frac{1}{\mu' - \lambda}$, $\mu' = \frac{1}{E[Y]} + \lambda$. For the slack time reallocation in STEP2, we give activity $1, 2, ..., n$ the same slack times as activity t because each path in the AND is performed independently.

OR Control Structure:

We just consider an OR control structure with a single activity per path by the same reason as an AND. In Figure 9, an OR control structure is transformed into an activity during STEP1. The average execution time of activity i is

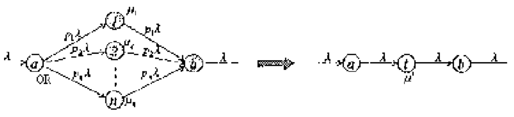


Figure 9. Transformation of an OR control structure

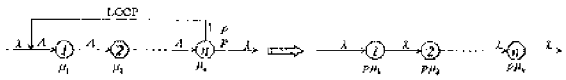


Figure 10. Transformation of a LOOP control structure

$W_i = \frac{1}{\mu_i - P_i \lambda}$ with the service rate μ_i and arrival rate $P_i \lambda$. When Y is a random variable on the average execution time of the OR, Y has a hyper-exponential distribution with the probability density function $f_Y(t) = \sum_{i=1}^{n} \frac{P_i}{W_i} e^{-\frac{t}{W_i}}$. We can finally obtain the service rate μ' of activity t from the average execution time of the OR, $E[Y] = \sum_{i=1}^{n} P_i W_i$. Because of $E[Y] = \frac{1}{\mu' - \lambda}$, $\mu' = \frac{1}{E[Y]} + \lambda$. Under STEP2, the slack time assigned to activity t is equally allocated to activity 1, 2, ..., n.

LOOP Control Structure:

Since each activity in a LOOP control structure is considered as an independent M/M/1 as mentioned in section 2, we can transform a LOOP into a sequence in STEP1. Because of the feedback in Figure 10, the arrival rate Λ of the LOOP is stated as: $\Lambda = \frac{\lambda}{p}$ from $\Lambda = \lambda + (1 - p)\Lambda$.

If $\rho_1, \rho_2, ..., \rho_n$ are the average number of service requests staying at activity 1, 2, ..., n, respectively, $\rho_i = \frac{\lambda}{p\mu_i}$, $i = 1, ..., n$. Hence, the average execution time $E[R]$ of the LOOP is $E[R] = \left(\frac{\rho_1}{1-\rho_1} + \frac{\rho_2}{1-\rho_2} + \ ... \ + \frac{\rho_n}{1-\rho_n}\right)\frac{1}{\lambda}$ $= \frac{1}{p\mu_1 - \lambda} + \frac{1}{p\mu_2 - \lambda} + ... + \frac{1}{p\mu_n - \lambda}$, which is equal to the average execution time of the sequence composed of n activities with the arrival rate λ and service rate $p\mu_i$ of activity i as in Figure 10. A LOOP control structure can, therefore, be transformed in a sequential control structure. In addition, Figure 11 shows that our transformation is correct. During STEP2, the slack time of an activity in the sequence is equally allocated to its corresponding activity in the LOOP.

5. Experiments

When determining the activity deadline, the PST method is based on the average execution time in an activity, which may implicitly consider the workload at the activity. This means that we analyzed and reflected workflow bottlenecks

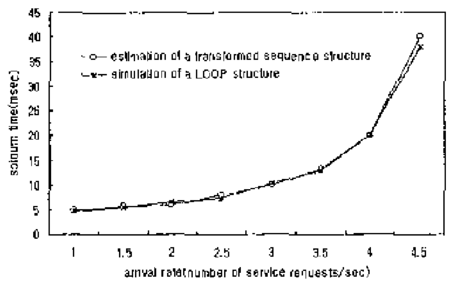


Figure 11. Estimation of transforming a LOOP to a sequence

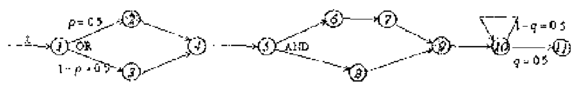
in the activity deadline assignment. In the experiments, we compare our PST with four previous methods, UD, ED, EQS, and EQF after slightly adjusting EQS and EQF to the static deadline assignment method without the information loss. Figure 12 is an example determining static activity deadlines by the policies of UD, ED, EQS, EQF, and our PST, based on the workflow schema. In this example, UD, ED, EQS, and EQF consider the average iteration of activity 10 for the LOOP, i.e., two iterations.

In Figure 13 and 14, activity deadlines determined by the PST method are all hard deadlines which means that all workflow instances will be escalated if they cannot meet the deadlines. Figure 13 shows the number of escalated activities for each deadline assignment method. We can notice that UD, ED, EQS, and EQF compared to PST may escalate workflow instances after many workflow activities have already been completed, which may require high escalation costs. Figure 14 depicts workflow throughput, the number of completed workflow instances satisfying their deadlines, for each deadline assignment method. Our PST method can support higher throughput than other methods in most cases with less escalation.

Table 1. Workflow Throughput according to Escalation Policies

Arrival rate	# of workflow instances satisfying hard deadlines	# of workflow instances satisfying hard/soft deadlines
2	8005	8322
4	7706	8109
6	7614	7980
8	7517	7928
10	4059	6644

On the other hand, Table 1 shows the number of workflow instances completed timely of 10000 workflow instances, i.e., workflow throughput under the two different escalation policies about activity deadlines in the PST method. One is that all activity deadlines are considered as hard deadlines, so all workflow instances not to meet activity deadlines are escalated. The other is that activity deadlines are divided into hard and soft deadlines using the concept of critical and non-critical activities mentioned in section 3. In case of soft activity deadlines, workflow instances can continue to perform their works even if they miss soft activity deadlines. The escalation policy distinguishing between hard and soft activity deadlines can somewhat reduce the possibility that some workflow instances being completed timely in the long run may be escalated due to missing the soft activity deadline. Consequently, workflow throughput can be improved if more appropriate escalation policies are applied.



Activity	Service Time	Deadline(UD)	Deadline(ED)
1	0.05 *sec*	5 *sec*	3.85 *sec*
2	0.2 *sec*	5 *sec*	4.05 *sec*
3	0.1 *sec*	5 *sec*	4.05 *sec*
4	0.2 *sec*	5 *sec*	4.25 *sec*
5	0.05 *sec*	5 *sec*	4.3 *sec*
6	0.1 *sec*	5 *sec*	4.4 *sec*
7	0.1 *sec*	5 *sec*	4.5 *sec*
8	0.1 *sec*	5 *sec*	4.5 *sec*
9	0.2 *sec*	5 *sec*	4.7 *sec*
10	0.1 *sec*	5 *sec*	4.8 *sec*
11	0.2 *sec*	5 *sec*	5 *sec*

Activity	Deadline(EQS)	Deadline(EQF)	Deadline(PST)
1	0.461 *sec*	0.192 *sec*	0.1 *sec*
2	1.072 *sec*	0.961 *sec*	0.561 *sec*
3	1.072 *sec*	0.961 *sec*	0.561 *sec*
4	1.683 *sec*	1.73 *sec*	1.553 *sec*
5	2.144 *sec*	1.922 *sec*	1.653 *sec*
6	2.655 *sec*	2.307 *sec*	1.888 *sec*
7	3.166 *sec*	2.692 *sec*	2.123 *sec*
8	3.166 *sec*	2.692 *sec*	2.123 *sec*
9	3.777 *sec*	3.461 *sec*	3.115 *sec*
10	4.388 *sec*	4.23 *sec*	4.007 *sec*
11	5 *sec*	5 *sec*	5 *sec*

* Workflow Deadline = 5 *sec* * Workflow Slack Time = 3.8 *sec*

Figure 12. Activity Deadline Allocation

6. Conclusion

Many business processes have time properties, which requires new time management technologies appropriate to workflow system environments. One of the most fundamental time management in a workflow is to manage workflow and activity deadlines. Because the workflow deadline is hardly changed in general, the way to assign activity deadlines effectively has been studied in many materials with the objective of improving workflow throughput and performance. Recently, the explosive increase of workflow users

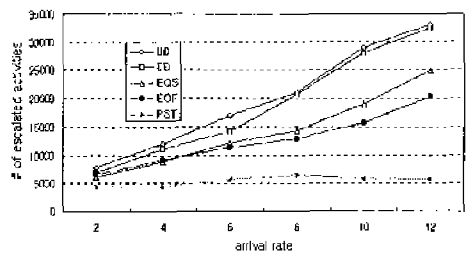


Figure 13. The number of escalated activities

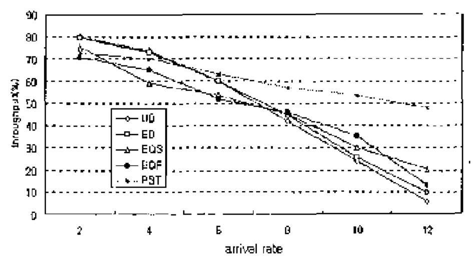


Figure 14. Workflow throughput

makes this issue more important.

In this paper we propose an effective scheme to improve workflow throughput by managing activity deadlines. First, we divides activities into two groups, i.e., a critical and non-critical activity group using the ICSF algorithm to find out the critical path in a workflow. The PST algorithm then determines static activity deadlines using queueing theory. If an activity belongs to a critical or non-critical activity group, it has the hard or soft deadline, respectively. According to the activity group, we can apply various escalation policies to improve workflow throughput. The usefulness of our scheme was shown through various experiments. For the further work, the escalation cost model appropriate to workflow system environments needs to be defined.

References

[1] R. L. Disney. Queueing networks. *American Mathematical Society Proceedings of Symposium in Applied Mathematics*, 1981.

[2] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. *Conference on Advanced Information Systems Engineering*, pages 286–300, 1999.

[3] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *In Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 428–437, 1993.

[4] P. Lawrence. *Workflow Handbook 1997*. John Wiley & Sons Ltd, 1997.

[5] E. Panagos and M. Rabinovich. Predictive workflow management. *The 3th International Workshop on NGITS*, 1997.

[6] E. Panagos and M. Rabinovich. Reducing escalation-related costs in wfmss. *In NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, pages 106–128, 1997.

[7] J. H. Son and M. H. Kim. Improving the performance of time-constrained workflow processing. *Journal of Systems and Software*, appeared.

[8] R. W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, 1989.