International Journal of Cooperative Information Systems
Vol. 10, No. 4 (2001) 509–530
© World Scientific Publishing Company

DEADLINE ALLOCATION IN A TIME-CONSTRAINED WORKFLOW

JIN HYUN SON, JUNG HOON KIM and MYOUNG HO KIM
Division of Computer Science
Department of Electrical Engineering and Computer Science,
KAIST 373-1 Kusong-dong Yusong-gu Taejon 305-701 Korea
E-mail: {jhson, kimjh, mhkim}@dbserver.kaist.ac.kr

Workflow management systems should efficiently manage workflow constraints such as time, resource, and cost during workflow executions. Especially, workflow time management is important in timely scheduling of workflow process execution, avoiding deadline violations, and improving the workflow throughput. Though there have been some studies on the dynamic deadline management in a workflow, the importance of the static deadline management has not been much addressed in the past.

We first describe our workflow model considered in this paper. Then, we propose a static deadline allocation method that can facilitate an efficient workflow processing. The proposed method can achieve high workflow throughput by analyzing the interrelated workflow components. We also present various experimental results that show the usefulness and efficiency of the method.

Keywords: Workflow, time management, deadline allocation, workflow management, queuing network.

1. Introduction

The concept of a workflow has recently accelerated the computerization and automation of business processes. A workflow, a highly abstracted business process, is a collection of activities interconnected by serial, alternative, parallel, or other workflow control structures. These control structures generate many execution paths in a workflow. An activity is called manual or automatic when humans are involved in its execution or software processes wholly perform its role, respectively. Here, humans or software processes to perform the role of an activity are called agents. A workflow instance denotes an actual business process in execution for a workflow. A workflow management system (WFMS) completely defines, manages and processes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.[1,2]

Since most business processes have time constraints in practice, time management is an important issue in workflow processing.[1,3] When a workflow instance violates the activity deadline during workflow execution, workflow management systems may escalate.[4,5] Escalation is defined as an automated or manual procedure that is invoked if a particular constraint or condition is not met.[2] In general

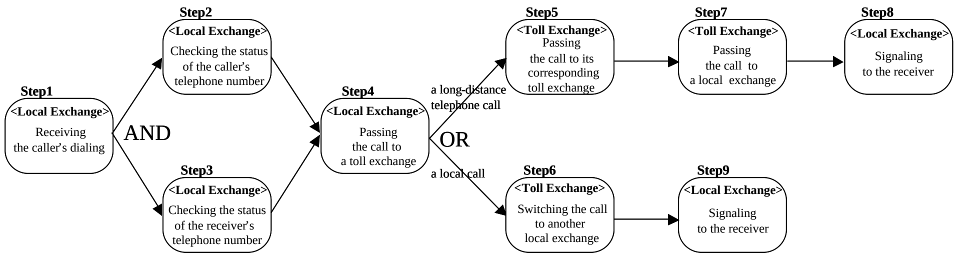

Step1
<Local Exchange>
Receiving the caller's dialing
AND
Step2
<Local Exchange>
Checking the status of the caller's telephone number
Step3
<Local Exchange>
Checking the status of the receiver's telephone number
Step4
<Local Exchange>
Passing the call to a toll exchange
OR
a long-distance telephone call
a local call
Step5
<Toll Exchange>
Passing the call to its corresponding toll exchange
Step6
<Toll Exchange>
Switching the call to another local exchange
Step7
<Toll Exchange>
Passing the call to a local exchange
Step8
<Local Exchange>
Signaling to the receiver
Step9
<Local Exchange>
Signaling to the receiver

Fig. 1. A telephone call process.

the effects of an escalation depend on the semantics of the activity that missed its deadline and, often, the execution of additional activities, the compensation of finished activities, or human intervention is required to proceed.[6,7] Because escalations may cause additional costs considerably, the number of escalated workflow instances needs to be reduced for efficient workflow processing.

Figure 1 depicts a simple example for a telephone call process with deadlines. A local exchange is an interface to the caller and the receiver of a telephone call, and a toll exchange is a mediator connecting two local exchanges or passing the telephone call to another toll exchange. In a district, there are a toll exchange and several local exchanges managed by the toll exchange. Each local exchange covers some areas in the district. The telephone call process has its overall completion time as the workflow deadline, i.e. from making a phone call to starting a conversation. And, each step has the activity deadline such as the maximum dialing time in Step 1, computing time in Step 2 and 3, and waiting time for the receiver in Step 8 and 9. Because telephone lines between a local exchange and a toll exchange or among toll exchanges are limited, Step 4, 5, 6, and 7 have the maximum waiting time for the connection as their activity deadlines. Escalations occurred in the telephone call process may give a message such as "dialing is too late" and "all lines are busy", and generate some information which can be utilized as statistics.

Especially, when a workflow applies to real-time applications such as the intelligent network services, the mobile telecommunication system, the 1-800 telephone service in the United States, and Web-based electronic commerce services mentioned in Ref. 8, workflow time management such as deadline is more critical in connection with high workflow throughput. If the activity deadlines can be determined properly, we can improve workflow throughput, i.e. the number of workflow instances that are timely completed.

In this paper, we first specify a workflow model considered in the paper. And then, we propose our deadline allocation method called Based on Queuing Network (BQN). Various experimental results that show the usefulness and efficiency of the method are finally presented.

The remainder of the paper is organized as follows: we explain our motivation for this topic in Sec. 2 and discuss related work in Sec. 3. Section 4 describes the workflow model considered in this paper. Section 5 presents our static deadline allocation method, and Sec. 6 shows in-depth analyses of the method with various experimental results. Some considerations are discussed in Sec. 7. Finally, we conclude our paper with its contribution and further works in Sec. 8.

2. Motivation

When business processes are shaped into time-constrained workflows, the activity deadlines are usually specified from the pre-defined workflow deadline during workflow build-time, while considering many internal workflow contexts such as workflow control flows and the expected execution time in an activity. These activity deadlines are called static activity deadlines compared to dynamic activity deadlines which are determined during workflow run-time. In general, the dynamic activity deadlines are computed based on the static activity deadlines. If a workflow instance misses the activity deadline, escalation may occur by workflow management systems. Otherwise, the difference between the activity deadline and its actual execution time, called slack time, will be appropriately distributed into all succeeding activities, resulting in dynamic activity deadlines. As you can expect, the dynamic activity deadline consisting of the static activity deadline and additional slack time assigned to the activity during workflow execution can reflect the up-to-date state of the workflow system.[6]

Although dynamic deadline allocation methods have been discussed in the literature,[6] the following two reasons clarify the necessity of an efficient static deadline allocation. First, dynamic methods would impose considerable amount of workloads on workflow management systems in many cases. After an activity completes its role, the available slack is distributed among all successors of the activity and the dynamic deadlines for its immediate successors are computed based on the information of other successors.[6] Hence, we need to analyze the control flows of a workflow to know the successors and their information, which may be more complex when the workflow includes a LOOP control structure. Because of that, some previous works consider only a well-structured workflow schema with no loops.[6,9] In addition, each workflow instance should have its own data structure to keep the dynamic deadline for each activity. Thus, workflow management systems especially to support large-scale workflow applications may have significant overheads for computing dynamic activity deadlines.

Next, the effectiveness of dynamic deadline allocation methods is considerably affected by the static deadlines. As we mentioned, the dynamic deadline of an activity is determined by its static deadline and additional slack time. Because the static deadline compared to the slack time generally holds a large portion of the dynamic deadline, it is important to effectively determine the static activity deadlines. As an example, Fig. 2 depicts a part of a workflow in which the expected execution time and the static deadline are specified for each activity. We assume that the dynamic activity deadlines are determined by PEX (Proportional Execution), one of the methods proposed in Ref. 6. If a workflow instance completed activity A_i

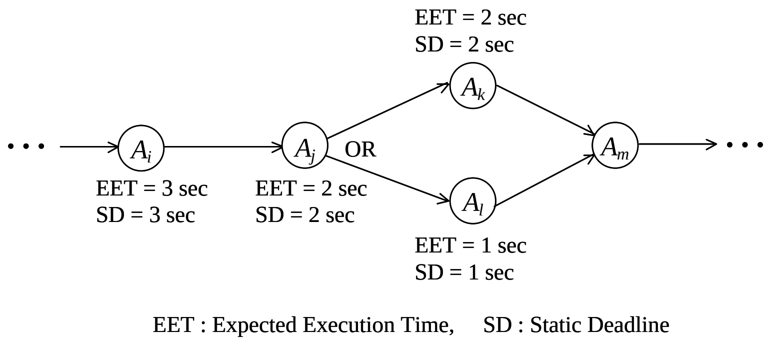


Fig. 2. Dynamic deadline allocation.

in 2 seconds, one second slack time will be distributed into all successors of activity A_i, i.e. A_j, A_k, A_l, and so on. Note that PEX eventually gives activity A_j and A_k the same slack time because their expected execution times are equal. This is not equitable in that the workload of activity A_k is less than that of activity A_j by the OR workflow control structure. Even though we select other dynamic deadline allocation methods proposed in Ref. 6, similar problems may occur. In this regard, we propose an efficient static deadline allocation method which not only gives less overhead to workflow management systems but also complements dynamic deadline allocation methods, resulting in promising high workflow throughput.

3. Related Work

Most existing workflow management systems are typically based on the client/server architecture that uses a centralized workflow engine. Because the centralized workflow management system has limitations in scalability, availability and reliability, the trend of workflow architecture has changed its focus toward distributed architectures.[10,11] For the high degree of parallel processing in distributed workflows, Ref. 12 has addressed the issue of task scheduling since there can be many workflow instances at the same time.

Improving workflow performance or throughput becomes a newly upcoming attractive goal especially in production workflows.[13,14] We can attain this goal by minimizing the number of escalated workflow instances in time-constrained workflows. In line with this, early escalation has been studied with the idea that it is preferable to escalate workflow instances as early as possible if the escalation is unavoidable.[7] And, Ref. 11 proposed a method to select and locate workflow servers in the appropriate subnets to manage large number of users and high volume data transfer within a workflow management system, which can reduce significantly the workflow server's workload. The workflow task allocation problem has been discussed in Ref. 15 with an objective of minimizing remote task calls as well as remote resource accesses. Reference 9 mentioned several workflow time management issues

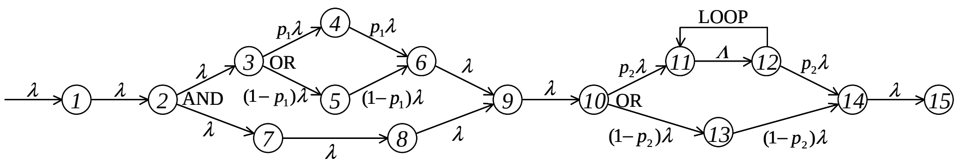


Fig. 3. Workflow queuing network.

such as computing activity deadlines, checking time constraints, and monitoring escalations by extending PERT.

Workflow management systems allow to define the activity deadlines in a workflow definition.[2] Originally, the deadline assignment problem has been intensively studied in distributed real-time systems in which a task is composed of subtasks.[16,17] Reference 16 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 computed as the deadline of the task minus the total expected execution time of its subsequent subtasks. These two methods determine activity deadlines statically. On the other hand, EQS and EQF divide the total remaining slack among the following subtasks equally and in proportion to their execution times, respectively. Because a task-subtasks relationship in distributed real-time systems and a workflow-activities relationship in workflow management systems have many similarities, these four deadline assignment methods can be partially applied to a workflow.[6,7] In addition, Ref. 6 proposed four dynamic deadline allocation methods in the context of a workflow: Total Slack (TSL), Proportional Execution (PEX), Proportional Escalation (PES) and Proportional Load (PLO). Under TSL, the available slack is assigned to each activity that is going to be executed next. On the other hand, PEX, PES, and PLO distribute the available slack among the succeeding activities in proportion to their average execution times, their escalation costs, and their loads, respectively.

4. Workflow Model

When a workflow designer defines workflows without any discipline for workflow control structures, plausible semantic and structural errors difficult to be found and corrected can be apt to be generated.[6,18] For example, if the destination of the LOOP's feedback in Fig. 3 is activity 6 instead of activity 11, the workflow does not have meaningful control flows due to the context of an AND control structure. In this regard, we define a well-formed workflow structure called a nested workflow. From now on, a nested workflow is called simply a workflow if there is no ambiguity.

Definition 4.1. *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 4.2. *The outermost non-sequential control structure in a nested workflow is called a non-sequential control block, or simply* ***control block****.*

The workflow depicted in Fig. 3 is the nested workflow with two control blocks, i.e. an AND control block and an OR control block, which contain an OR control structure and a LOOP control structure, respectively.

Numerous natural physical and organic processes exhibit behavior that is probably meaningfully modeled by Poisson processes. An important application of the Poisson distribution arises in connection with the occurrences of events of a particular type over time. The exponential distribution is frequently used as a model for the distribution of times between the occurrence of successive events such as customers arriving at a service facility. Because of them, the Poisson process and the exponential distribution have been used to analyze many areas of computer engineering.[19] We also consider that the service requests for a workflow arrive by a Poisson process and each activity agent has exponential service time. As a result, we can model a workflow as a M/M/1 queuing network such as telephone networks or computer communication networks.

Assumption 4.1. *Queue discipline in an activity is first come-first served (FCFS), i.e. workflow instances are served in the same order in which they arrive.*

Assumption 4.2. *The queue size of an activity is sufficiently large to accommodate a large number of workflow instances. Namely, workflow instances can wait the service of an activity for a long time.*

Each activity in a M/M/1 workflow queuing network is known to be an independent M/M/1 queuing system.[20] Namely, the activity agent is modeled by a M/M/1 queuing system, i.e. a single-threaded agent. However, we should consider two cases: one is that the role of an activity can be supported by multi-threaded or multiple agents. The other is that agents in a workflow system may be able to execute activities belonging to different workflows. In reality, multi-threaded or multiple agents may be able to work on several workflow instances at the same time. Thus, a multi-threaded agent is modeled by a M/M/c, and multiple agents are modeled by y number of M/M/c, where c is the number of threads in an agent and y is the number of duplicated agents. When an agent supports activities belonging to different workflows, it is not directly modeled by a M/M/1 because its service rate may differ depending on the activity executed next, which makes the computation of the average execution time at the agent difficult. To support these two cases, we propose transformation methods from them to a single-threaded M/M/1 agent, which will be discussed in Sec. 7.1.

In a M/M/1 queuing network, we can specify the arrival and departure process in each activity as in Fig. 3 from the information of the initial arrival rate to the workflow, the service rate of each activity agent, and the branch probability in an OR and a LOOP. It is based on the following facts and the property of time reversibility.[20]

It is known that the departure process of a M/M/1 is a Poisson process if its arrival process is a Poisson process.[19] Hence, all sequential activities in a M/M/1 workflow queuing network have Poisson arrival and departure processes. And, the decomposition and superposition of independent Poisson processes in the OR control structure, e.g. activities 10 and 14 of Fig. 3 are known to be also Poisson processes.[20] In case of the AND control structure, the arrival and departure process of an AND split activity, e.g. activity 2 are clearly Poisson processes,[19] 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. We can, however, assume that the arrival process of an AND join activity is a Poisson process because of the following reason. A workflow instance at an AND split activity with n branches is divided into n sub-requests which run in parallel. These n sub-requests must be synchronized at an AND join activity. The synchronization is dependent on the lastly arrived sub-request among n sub-requests at the AND join activity. It means that a point of synchronized time is greatly determined by the path with the longest average execution time of n branches. Thus, the arrival process of an AND join activity can be approximated by behaviors of the requests going through the longest execution path. In view of these facts, we can assume that the arrival process of an AND join activity is a Poisson process. A LOOP control structure has any kind of feedback, that is, service requests can return to previously visited activities. Even though the actual internal flow of a LOOP is not a Poisson process due to its feedback, it is known that the LOOP control structure behaves as if its activities are independent M/M/1 systems.[21] Hence, a LOOP control structure can also be a part of a M/M/1 queuing network. As a result, a M/M/1 queuing network in which each activity has Poisson arrival and departure processes as in Fig. 3 can model a nested workflow.

5. Deadline Allocation Method

Since there are many workflow instances executed concurrently, some workflow instances should 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 at an activity is *the average service time* of the activity agent plus *the average waiting time* of the workflow instance at the queue of the activity. Therefore, our proposed deadline allocation method, called BQN, is based on the average execution time in an activity.

Workflow deadlines in general are affected by the requirement of real-world business process as well as the expected execution time of each activity. When "the critical path" is defined as "a path with the longest execution time" in a workflow, it can be an important guideline in deciding the workflow deadline. Thus, in our paper we consider that the workflow deadline is determined by the longest execution time of the critical path plus certain amount of time required by the target business process. And then, the static activity deadlines are computed based on the workflow deadline by our BQN method. If the workflow deadline is determined to be less than the longest execution time of the critical path, a large portion of workflow instances may miss the workflow deadline and be finally escalated in many cases. The concept of the critical path will be discussed in Sec. 7.2.

Definition 5.1. *The workflow slack time* (W_{st}) *is the workflow deadline minus the longest average service time among many execution paths in the workflow.*

Definition 5.2. *The time assigned to activity i from the workflow slack time is called the static slack time of the activity* (A^i_{st}).

Definition 5.3. *The static deadline of activity i is the average service time of activity i plus the static slack time assigned to activity i.*

The workflow slack time is divided into the static slack times of activities in proportion to their average execution times, i.e. $W_{st} = \sum_{i=1}^{n} A^i_{st}$. Definition 5.3 intimates that BQN that determines the static deadline of an activity by giving it extra slack time besides its pre-defined average service time. Hence, the main goal of BQN is to determine the static slack time assigned to each activity, which will be explained in the next section. Because the workflow deadline is determined based on the critical path in the paper, the static slack time assigned to an activity may include the average waiting time at the activity. This means that the static deadline of an activity will be larger than its average execution time.

5.1. *BQN algorithm*

The BQN algorithm is composed of two stages, STAGE1 and STAGE2. STAGE1 first transforms a nested workflow into a sequential workflow only composed of a sequence control structure. With this sequential workflow, we divide the workflow slack time among activities in proportion to their average execution times in STAGE2, resulting in determining the static activity deadlines.

Figure 4 shows an example to explain our BQN method. Note that the result of STAGE1 is a sequential workflow and the process of STAGE2 is in the reverse order of STAGE1. To get a sequential workflow from a nested workflow during STAGE1, we should substitute a control block with an activity. There are two control blocks in Fig. 4, an AND control block and an OR control block. These control blocks are replaced with activity $T1$ and $T2$, respectively, by STAGE1. When substituting a control block with an activity, we start from transforming its innermost control structure into an activity and end with transforming its outermost control structure into an activity. For example, in the AND control block in Fig. 4, its innermost OR control structure including activities 4 and 5 are first transformed into activity $R1$. And then, two sequence control structures are transformed into activity $S1$ and $S2$, respectively. Finally, we can get activity $T1$ substituting the AND control block. The OR control block in Fig. 4 can also be substituted by activity $T2$ in the same process as the AND control block. As a result, we can obtain a sequential workflow from a nested workflow by STAGE1 of the BQN method.

With this sequential workflow, we distribute the workflow slack time among activities in proportion to their average execution times in STAGE2. Thus, each activity in the sequential workflow is assigned its static slack time. If an activity, i.e. activity $T1$ or $T2$ in Fig. 4 replaces a control block of the nested workflow, we should redistribute the static slack time assigned to the activity to its corresponding activities, but we need not do it for other activities, i.e. activities 1, 2, 9, 10, 14, and 15 because they belong to the original nested workflow. The redistribution of the static activity slack time is proceeded in the reverse order of STAGE1. After STAGE2, all activities constituting the original nested workflow can have their own static slack times and are finally assigned their static deadlines.

The algorithm of BQN is described in Fig. 5. The computational complexity of STAGE1 in the BQN algorithm is $O(cd)$ and that of STAGE2 is $O(n)$, where c is the number of control blocks, d is the depth of the nesting in the control block, and n is the number of activities newly created during STAGE1. Because n is generally larger than cd, the computational complexity of the BQN algorithm is $O(n)$. Because the static activity deadlines are computed during workflow build-time, no computational overhead may be given to a workflow engine. From now, we discuss in detail the transformation process of STAGE1 and the redistribution process of STAGE2 for each workflow control structure, i.e. Sequence, AND, OR, and LOOP.

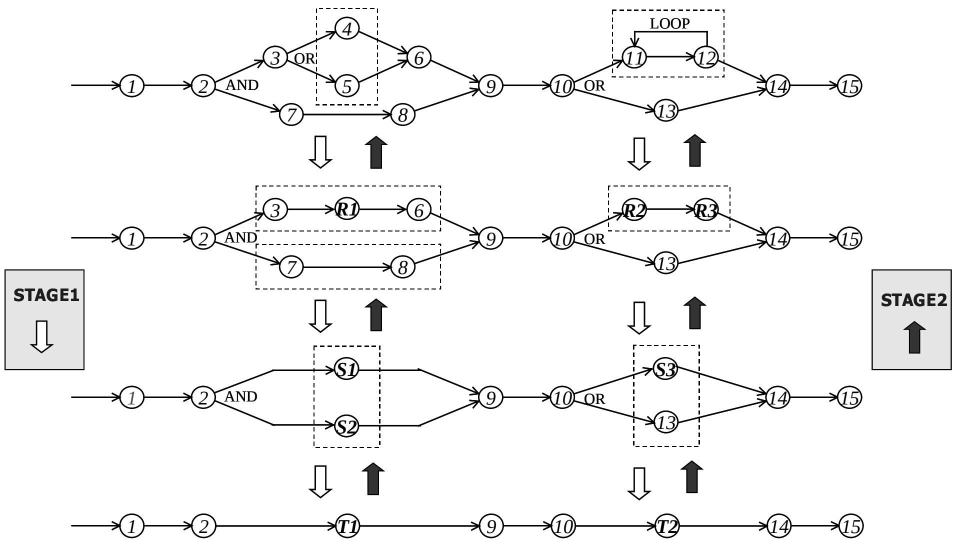


Fig. 4. An example of the BQN method.

```
STAGE1: Transform a nested workflow into a sequential workflow
        For (each control block) {
          DO {
            1. IF (its innermost control structure is a LOOP)
                 1.1 Transform the LOOP into a sequence
            2. ELSE IF (its innermost control structure is an AND or an OR)
                 2.1 In an AND or an OR consisting of $n$ paths, transform each sequential path into an activity
                 2.2 Transform the control structure into an activity
          } WHILE (the control block is not yet transformed into an activity)
        }
        /* We obtain a new sequential workflow. */

STAGE2: Redistribute the static activity slack times in the reverse order of STAGE1
        4. Distribute the workflow slack time to activities within the sequential workflow
           in proportion to their average execution times

        For (each activity that is assigned the static activity slack time and newly created during STAGE1) {
          5. IF (it substituted for an AND or an OR control structure)
               5.1 The static activity slack time is equally assigned to activities within the AND or the OR.
          6. ELSE IF (it substituted for a sequence control structure)
               6.1 The static activity slack time is distributed to activities within the sequence
                    in proportion to their average execution times.
          7. ELSE IF (it substituted for an activity within a LOOP)
                7.1 The static activity slack time is equally assigned to the activity within the LOOP.
        }
        /* Finally, the static activity deadlines are determined */
```

Fig. 5. BQN algorithm.

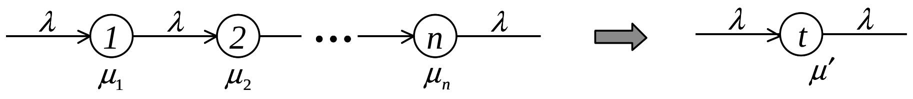


Fig. 6. Transformation of a sequence.

5.2. *Sequence control structure*

In STAGE1, a sequence control structure is transformed into an activity as in step 2.1 of the BQN algorithm. For the transformation of Fig. 6, we 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 in a M/M/1 is $W = \frac{1}{\mu-\lambda}$ 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 belonging to the sequence.[20] Hence, when the average execution time of activity $i(i = 1, \ldots, n)$

is $W_i = \frac{1}{\mu_i - \lambda}$, the service rate μ' of activity t can be obtained as:

$$\sum_{i=1}^{n} W_i = \frac{1}{\mu' - \lambda}$$

$$\mu' = \frac{1}{\sum_{i=1}^{n} W_i} + \lambda$$

When an activity substituted for a sequence control structure during STAGE1, the static slack time assigned to the activity is distributed into activities within the sequence in proportion to their average execution times as step 6 in the BQN algorithm. For example, the static slack time i_s assigned to activity $i(i = 1, \ldots, n)$ belonging to the sequence is $i_s = t_s * \frac{W_i}{\sum_{j=1}^{n} W_j}$, where t_s is the static slack time assigned to activity t and W_i is the average execution time of activity i in Fig. 6.

5.3. *AND control structure*

An AND control structure is transformed into an activity which has the same average execution time with it as in step 2.2 during STAGE1 of the BQN algorithm. Because each sequential path in an AND control structure is transformed into an activity in step 2.1 of the BQN algorithm using Sec. 4.2, we only consider an AND control structure with a single activity per path as in Fig. 7. Let random variables on the average execution times of activity $1, 2, \ldots, n$ in Fig. 7 be $X_1, X_2, \ldots, X_n$, respectively. Because the average execution time of activity i is $W_i = \frac{1}{\mu_i - \lambda}$ with its service rate μ_i, the probability density function $f_{X_i}(t)$ of X_i is

$$f_{X_i}(t) = 1 - e^{-(\mu_i - \lambda)t}, \qquad i = 1, \ldots n$$

When Y is a random variable on the average execution time of the AND control structure, Y is $\max(X_1, X_2, \ldots, X_n)$ and the cumulative distribution function $G_Y(x)$ is

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

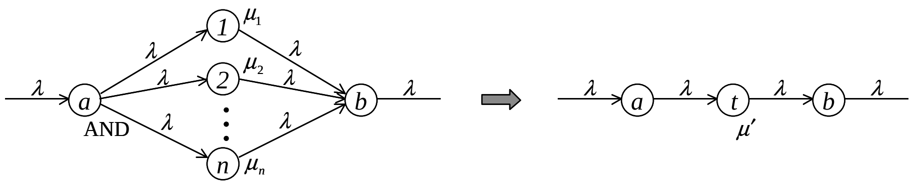


Fig. 7. Transformation of an AND control structure.

Because the average execution times of activity t and the AND control structure in Fig. 7 are $W_t = \frac{1}{\mu' - \lambda}$ and $E[Y] = \int_0^\infty 1 - G_Y(x)dx$, respectively, we can determine the service rate μ' of activity t corresponding to the AND as follows:

$$E[Y] = \frac{1}{\mu' - \lambda}$$

$$\mu' = \frac{1}{E[Y]} + \lambda$$

On the other hand, the static slack time assigned to activity t of Fig. 7 is redistributed to activities within its corresponding AND control structure as mentioned in step 5 of the BQN algorithm during STAGE2. For this, we give activity $1, 2, \ldots, n$ of Fig. 7 the same slack time as activity t since activity $1, 2, \ldots, n$ are executed concurrently and independently. And then, their deadlines are reset to $\mathrm{MAX}(A^1_{st}, A^2_{st}, \ldots, A^n_{st})$, where A^i_{st} is the static deadline of activity i, because their executions should be synchronized at activity b. Here, the static slack time of activity $i (i = 1, \ldots, n)$ is recomputed as ($\mathrm{MAX}(A^1_{st}, A^2_{st}, \ldots, A^n_{st})$ − *the service time of activity* i) from Definition 5.3. After that, the redistribution of the static slack time assigned to the activity uses its new static slack time.

5.4. *OR control structure*

During STAGE1, an OR control structure is transformed into an activity in step 2.2 of the BQN algorithm. We can also consider an OR control structure with a single activity per path by the same reason as an AND control structure. When Y is a random variable on the average execution time of the OR control structure, Y has a hyper-exponential distribution with the probability density function $f_Y(t) = \sum_{i=1}^n (\frac{p_i}{W_i})e^{-\frac{t}{W_i}}$, where $W_i = \frac{1}{\mu_i - p_i\lambda}$ is the average execution time of activity $i (i = 1, \ldots, n)$ with the service rate μ_i and the arrival rate $p_i\lambda$ in Fig. 8. Because the average execution times of the OR and activity t in Fig. 8 are $E[Y] = \sum_{i=1}^n p_i W_i$ and $W_t = \frac{1}{\mu' - \lambda}$, respectively, we can finally obtain the service rate μ' of activity t as follows:

$$E[Y] = \frac{1}{\mu' - \lambda}$$

$$\mu' = \frac{1}{E[Y]} + \lambda$$

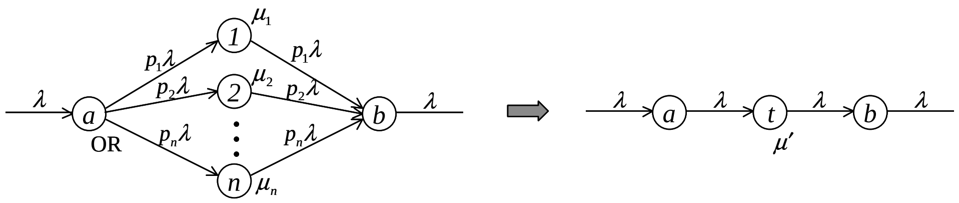


Fig. 8. Transformation of an OR control structure.

For step 5 of STAGE2 in the BQN algorithm, the static slack time assigned to activity t is equally assigned to activity 1, 2,..., n belonging to the OR control structure since they are performed exclusively.

5.5. *LOOP control structure*

Since each activity in a LOOP control structure is an independent M/M/1 as mentioned in Sec. 4, the LOOP can be transformed into a sequence as in Fig. 9, which comes under step 1 of STAGE1 in the BQN algorithm. Because the arrival rate Λ of the LOOP is the sum of its original arrival rate λ and its feedback arrival rate $(1-p)\Lambda$, the arrival rate Λ of the LOOP is stated as:

$$\Lambda = \lambda + (1-p)\Lambda$$

$$\Lambda = \frac{\lambda}{p}$$

where $1-p$ is the probability that the feedback occurs in activity n.

If $\frac{\rho_i}{1-\rho_i}$ is the average number of service requests waiting in front of activity i in Fig. 9, respectively, then ρ_i becomes $\frac{\lambda}{p\mu_i}, i = 1,\ldots,n$. Hence, the average execution

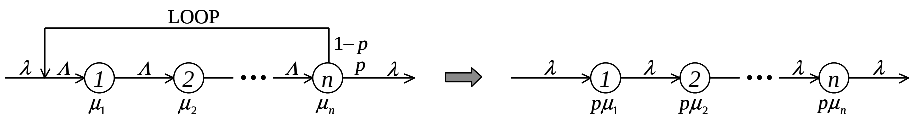


Fig. 9. Transformation of a LOOP control structure.

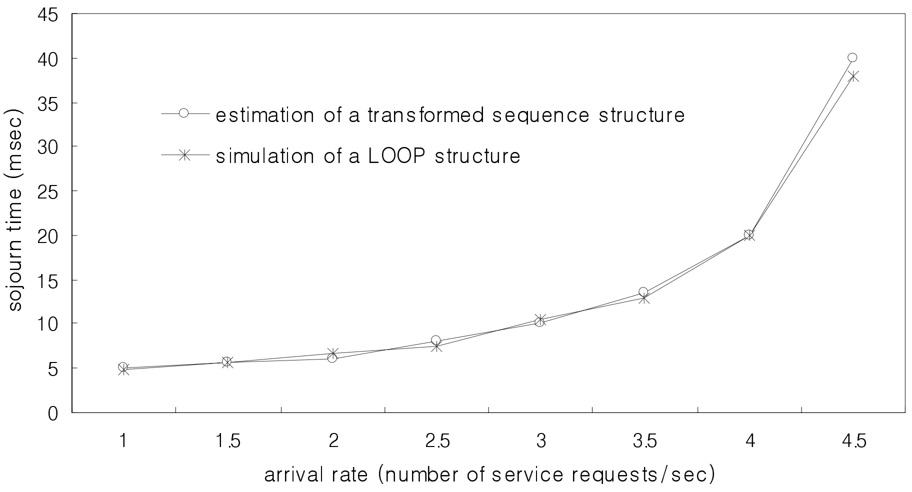


Fig. 10. Estimation of transforming a LOOP to a sequence.

time $E[R]$ of the LOOP is

$$E[R] = \left(\frac{\rho_1}{1-\rho_1} + \frac{\rho_2}{1-\rho_2} + \ldots + \frac{\rho_n}{1-\rho_n}\right)\frac{1}{\lambda}$$

$$= \frac{1}{p\mu_1 - \lambda} + \frac{1}{p\mu_2 - \lambda} + \ldots + \frac{1}{p\mu_n - \lambda}$$

which is equal to the average execution time of the sequence composed of n activities with the arrival rate λ and the service rate $p\mu_i$ as in Fig. 9. A LOOP control structure can, therefore, be transformed into a sequence control structure. In addition, the experimental result of Fig. 10 shows that our transformation is reasonable.

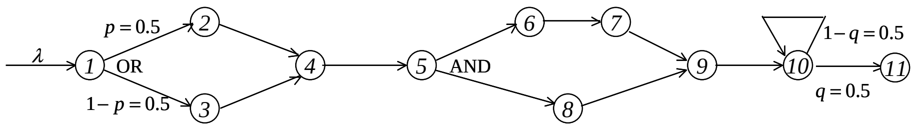
On the other hand, the static slack time of an activity within the sequence is equally assigned to its corresponding activity in the LOOP for step 7 of STAGE2.

6. Experiments

We have done several experiments to evaluate deadline allocation methods using the Arena and DEVSim++. The Arena by Systems Modeling (SM) Corporation is a simulation tool that enables us to build models of business process reengineering, expert systems, computers, etc. and analyze those models. DEVSim++ is a C++-based discrete-event modeling framework which has been in many areas of systems' design such as communication network design and parallel computer architectures design.

The experiments have been done with two different purposes: one is to show that our static deadline allocation method, BQN, guarantees higher workflow throughput than previous static deadline allocation methods, UD and ED, proposed in distributed soft real-time systems.[16] The other is to prove that an efficient static deadline allocation method not only matches well but also complements dynamic deadline allocation methods for high workflow throughput. Figures 12, 13, and 14 are illustrated for the first purpose and Fig. 15 for the second purpose.

Figure 11 depicts a workflow used in the performance experiments and specifies the static activity deadlines determined by UD, ED, and our BQN methods with the assumption that the workflow deadline is 5 seconds. Note that UD and ED would give preceding activities enough static deadlines while BQN appropriately distributes the workflow slack time over the whole workflow. As you can expect, workflow instances may be escalated mostly at the latter activities of the workflow if the static activity deadlines are determined by UD and ED. BQN, however, may tend to escalate many workflow instances at preceding activities with the thought that they would eventually seem to miss the workflow deadline. We assume in the experiments that the escalated workflow instances get ignored without resubmission for execution and the effect of an escalation is to compensate finished activities. In other words, if a workflow instance does not meet the static deadline of activity 9 in Fig. 11, it is immediately removed from the workflow execution and the number of escalated activities is 7, i.e. the number of already finished activities.



Activity	Service Time	Deadline(UD)	Deadline(ED)	Deadline(BQN)
1	0.05 sec	5 sec	3.85 sec	0.1 sec
2	0.2 sec	5 sec	4.05 sec	0.561 sec
3	0.1 sec	5 sec	4.05 sec	0.561 sec
4	0.2 sec	5 sec	4.25 sec	1.553 sec
5	0.05 sec	5 sec	4.3 sec	1.653 sec
6	0.1 sec	5 sec	4.4 sec	1.888 sec
7	0.1 sec	5 sec	4.5 sec	2.123 sec
8	0.1 sec	5 sec	4.5 sec	2.123 sec
9	0.2 sec	5 sec	4.7 sec	3.115 sec
10	0.1 sec	5 sec	4.8 sec	4.007 sec
11	0.2 sec	5 sec	5 sec	5 sec

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

Fig. 11. Activity deadline allocation.

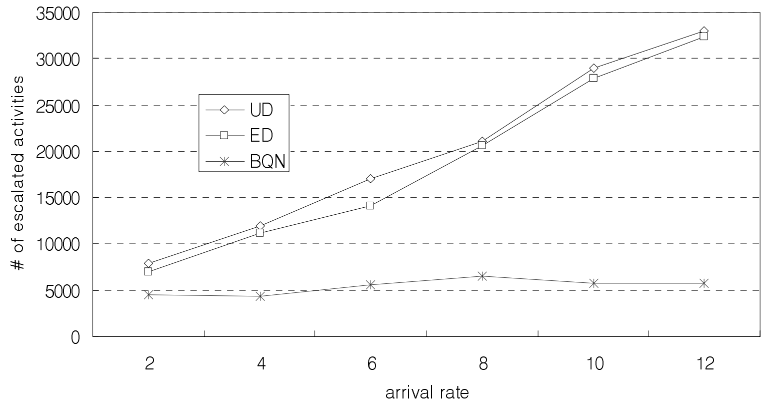


Fig. 12. The number of escalated activities.

Figure 12 shows the number of escalated activities by UD, ED, and BQN when 10,000 workflow instances are created by the arrival rate. We can notice that BQN greatly reduces the number of escalated activities, compared to UD and ED. On the other hand, BQN seems not to be affected by the arrival rate because the number of escalated activities by BQN is almost invariable. But, this is because BQN escalates a large portion of workflow instances in the preceding activities at the high arrival

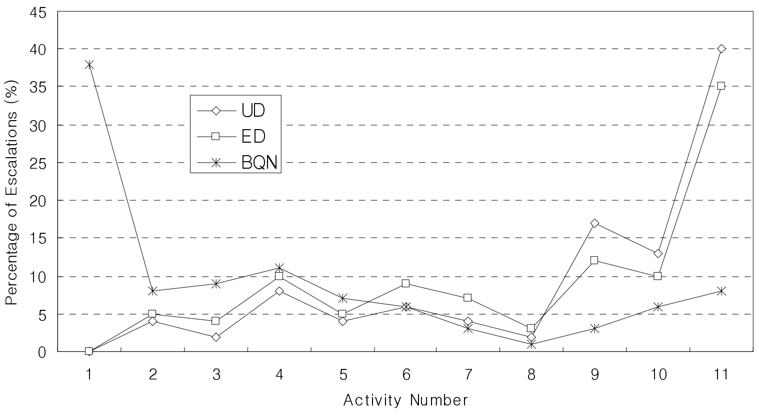


Fig. 13. Percentage of escalations.

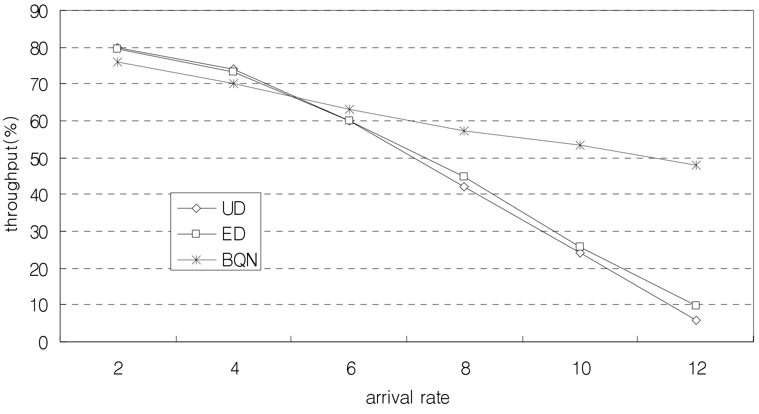


Fig. 14. Workflow throughput under static deadline allocation methods.

rate, not meaning that the number of completed workflow instances satisfying the workflow deadline is constant regardless of the arrival rate. For more detailed explanations of the behaviors of these methods, Fig. 13 illustrates the percentage of escalations for each activity under the arrival rate 6. According to expectation, the rate of workflow instances escalated by UD and ED is high in the latter activities while BQN escalates many workflow instances in the preceding activities. Figure 14 shows the workflow throughput for each method, i.e. the number of completed workflow instances out of the total number of instances submitted for execution. Our BQN guarantees high workflow throughput in many cases, especially, in the high arrival rate. Note that the workflow throughput of BQN is less than that of other methods at the arrival rates 2 and 4 in Fig. 14 even though the number of escalated activities by BQN is less than that of others at the same arrival rate in Fig. 12. The reason is based on the following facts. We have considered in the experiments the

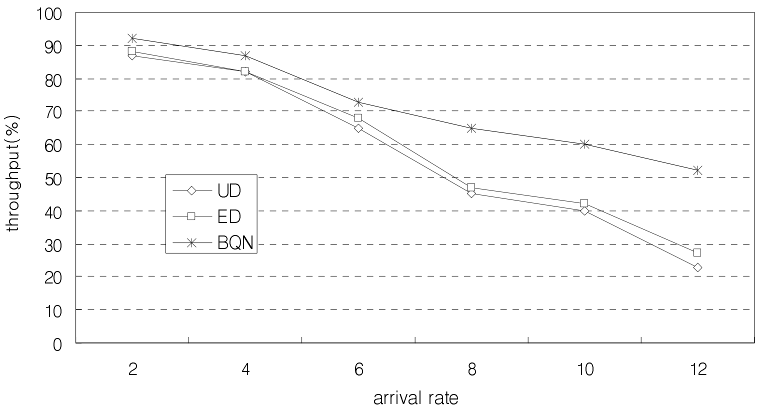


Fig. 15. Workflow throughput under a dynamic deadline allocation method (PEX).

effect of an escalation as the compensation of finished activities. Because UD and ED escalate relatively many workflow instances in the latter activities of a workflow, the number of escalated activities by them is very large compared to BQN as in Fig. 13 no matter how BQN does not support high workflow throughput in the arrival rates 2 and 4.

From the result of Fig. 15, we insist that an efficient static deadline allocation method is necessary even though dynamic deadline allocation methods are applied during workflow execution. When PEX, one of dynamic deadline allocation methods proposed in Ref. 6. adopts our BQN as its static deadline allocation method instead of UD and ED, we can accomplish higher workflow throughput as in Fig. 15. This means that our proposed method can match well and complement dynamic deadline allocation methods, which is one of our contribution in the paper.

7. Discussion

7.1. *M/M/*1 *activity agent*

We mentioned in Sec. 4 that a single-threaded M/M/1 agent in the paper models the activity agent. To comply with this, we discuss transformation methods for two cases: one is for multi-threaded or multiple agents, and the other is for agents to support activities belonging to different workflows.

Because multi-threaded or multiple agents may be able to work on several workflow instances at the same time, a multi-threaded agent is modeled by a M/M/c and multiple agents are modeled by y number of M/M/c, where c is the number of threads in an agent and y is the number of duplicated agents. Thus, it is necessary to transform them into a single-threaded agent, i.e. M/M/1. In this transformation, when W denotes the average execution time of the source queuing system and W' denotes that of the destination queuing system. The basic principle is to adjust the service rate of the destination queuing system while preserving the equality

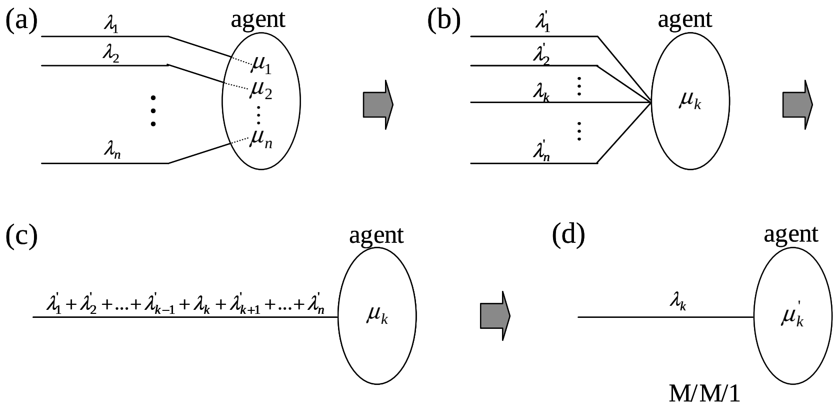


Fig. 16. Agent transformation.

$W = W'$. Namely, in the transformation from a M/M/c to a M/M/1, we can compute the service rate $\mu'(= \frac{1}{W} + \lambda)$ of the M/M/1 from the equation $W = W'$ ($W' = \frac{1}{\mu'-\lambda}$), where W is the average execution time of the M/M/c which is already known in the literature,[20] W' is that of the M/M/1, and λ is the arrival rate of the M/M/1. For multiple agents modeled by y number of M/M/c, they are first transformed to a M/M/cy and then finally to a M/M/1 with the similar method mentioned above. As a result, we can transform multi-threaded or multiple agents into a single-threaded M/M/1 agent.

The agent to support activities belonging to different workflows is not directly modeled by a M/M/1 because its service rate may differ depending on the activity executed next. We handle this problem with a transformation method from which the agent can approximately be modeled by a M/M/1. For an agent supporting n number of activities which belong to different workflows, let the arrival rate for activity $i(i = 1, \ldots, n)$ be λ_i and the service rate of the agent to support activity i be μ_i as in Fig. 16(a). When we want to analyze the average execution time of the agent in the aspect of activity k, we transform the model of Fig. 16(a) into a M/M/1 for activity k as Fig. 16(d). Because each activity $i(i \neq k)$ in Fig. 16(a) is an independent M/M/1 with the Poisson arrival rate λ_i and the exponential service rate μ_i, we first transform it into a virtual M/M/1 activity with the Poisson arrival rate λ_i' and the exponential service rate μ_k, while keeping the average execution time of the activity unchangeable. Hence, we can compute $\lambda_i' = \mu_k - \mu_i + \lambda_i$ ($i = 1, \ldots, n, i \neq k$) from the equation $\frac{1}{\mu_i - \lambda_i} = \frac{1}{\mu_k - \lambda_i'}$, where $\frac{1}{\mu_i - \lambda_i}$ is the average execution time of activity i and $\frac{1}{\mu_k - \lambda_i'}$ is that of the virtual activity transformed from activity i. As a result, Fig. 16(b) is generated with the service rate μ_k of the agent supporting activity $i(i = 1, \ldots, n)$ and the arrival rate λ_i' of activity $i(i \neq k)$. Next, we can easily obtain a M/M/1 of Fig. 16(c) with the arrival rate $\lambda_1' + \lambda_2' + \ldots + \lambda_{k-1}' + \lambda_k + \lambda_{k+1}' + \ldots + \lambda_n'$ and the service rate μ_k since the superposition of independent Poisson processes is known to be also a Poisson process.[20] Finally,

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

2. For (each control block) {
     3. IF (the innermost control structure in the control block is an AND or OR)
          3.1 Select the longest average execution path in the control structure.
     4. ELSE IF (the innermost control structure in the control block is a LOOP)
          4.1 Transform it into a sequence.
     } WHILE (a sub-critical path of the control block is not yet determined)
   }

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

Fig. 17. ICSF Algorithm.

the M/M/1 of Fig. 16(c) is transformed into a M/M/1 of Fig. 16(d) with the arrival rate λ_k and the service rate μ'_k. The service rate μ'_k can be computed from the equation $\frac{1}{\mu'_k - \lambda_k} = \frac{1}{\mu_k - (\lambda'_1 + \lambda'_2 + \ldots + \lambda'_{k-1} + \lambda_k + \lambda'_{k+1} + \ldots + \lambda'_n)}$ to preserve the average execution time at the agent equally. In consequence, we can get a new M/M/1 agent in the aspect of activity k which is transformed from an agent supporting activities that belong to different workflows.

7.2. *Critical path*

By means of workflow time management, a workflow instance may be escalated if it does not meet the activity deadline. Because of the escalation cost, it is necessary to make workflow instances escalated as less as possible. Note that there may be some workflow instances which are escalated due to violating the intermediate activity deadlines even though they can finally complete the workflow timely. In this section, we discuss a candidate method to handle this problem using the concept of the critical path.

The critical path has been addressed in many research areas, especially, PERT/CPM for the project management.[22] With the definition of "the critical path" as "the longest execution path", PERT/CPM cannot be used to determine the critical path in the nested workflow because it only considers the network as consisting of activities interconnected by sequence and parallel relationships.[22] So, we propose a simple method called the Innermost Control Structure First (ICSF) to find out the critical path. In the context of a workflow, the concept of the critical path can be effectively utilized in many workflow issues such as workflow resource management and workflow time management.

Figure 17 describes its algorithm. First, all sequential control structures except ones within control blocks are clearly part of the critical path because the sequences

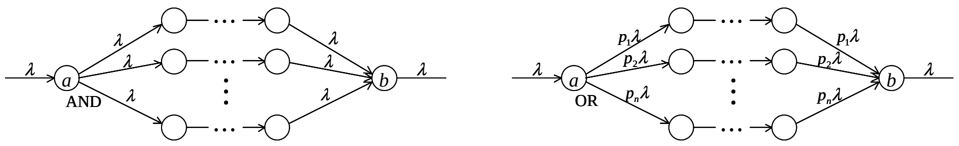


Fig. 18. AND or OR control structure.

are all unique in the workflow (step 1 in the ICSF algorithm). Next, ICSF determines a sub-critical path with the longest average execution time in each control block by selecting the longest execution path from the innermost control structure to the outermost control structure of the control block, which comes under step 2 in the ICSF algorithm. If the innermost control structure is an AND or OR, select a path with the longest average execution time (step 3). If the innermost control structure is a LOOP, transform it into a sequence control structure (step 4). Because the average execution time in a M/M/1 activity is $W = \frac{1}{\mu_i - \lambda_i}$ with the arrival rate λ_i and the service rate μ_i, the longest execution path in an AND or OR control structure as in Fig. 18 is a path whose total average execution time is $\mathrm{MAX}(\sum_i \frac{1}{\mu_i - \lambda_i})$, where λ_i is the arrival rate and μ_i is the service rate of activity i in the path. The transformation from a LOOP into a sequence is presented in Sec. 5.5. Finally, we can determine the critical path by combining all the sub-critical paths, which is for step 6 in the ICSF algorithm.

To reduce the possibility that workflow instances which can completely finish the whole workflow in time are escalated due to violating the intermediate activity deadlines, we distinguish between two kinds of activities, i.e. critical and non-critical, while applying different escalation policies. In the case of conditional executions, there may be more than one OR branches that can belong to the critical path or a path whose total execution time is within distance d from the critical path. If an activity belongs to either the critical path or OR branches within distance d from the critical path, we call it a critical activity; otherwise, a non-critical activity. The value of distance d may be decided by a workflow analyzer. We can expect that a workflow instance missing the deadline of any critical activity has very high possibility not to meet the workflow deadline finally. Hence, if a workflow instance does not meet the deadline of a critical activity, it may be escalated immediately, which means that the deadline is hard. On the other hand, the deadline of a non-critical activity is soft in the sense that we make a workflow instance continue to perform its execution if it does not exceed the activity deadline by α percent, where α is the threshold controlled by a workflow analyzer.

Table 1 proves the usefulness of our scheme that divides activity deadlines into two categories and applies different escalation policies for the purpose of improving workflow throughput. For the experiment, we assume that $d = 0$ and $\alpha = 20\%$. The number of timely completed workflow instances, i.e. workflow throughput is different according to the selected escalation policy as in Table 1. One is that all activities in a workflow have only hard deadlines, so all workflow instances not to

Table 1. Workflow throughput according to escalation policies.

Arrival rate	# of timely completed workflow instances in a workflow with only hard deadlines	# of timely completed workflow instances in a workflow with hard/soft deadlines
2	8005	8322
4	7706	8109
6	7614	7980
8	7517	7928
10	4059	6644

meet activity deadlines are immediately escalated. The other is that activity deadlines are divided into hard and soft using the concept of critical and non-critical activities. The escalation policy distinguishing between hard and soft activity deadlines can somewhat reduce the possibility that workflow instances being completed timely in the long run may be escalated due to missing the soft deadlines in any intermediate activities.

8. Conclusion

Business processes with time properties requires new time management technologies when they are abstracted to time-constrained workflows. One of the most fundamental time management in a workflow is to deal with workflow and activity deadlines. So far, the efficient methods to manage dynamic activity deadlines have been studied in some materials with the objective of improving workflow throughput. Recently, the explosive increase of workflow users makes this issue more important and requisite.

In this paper, we proposed an efficient static activity deadline allocation method called BQN which reflects both the execution time of an activity and the characteristics of workflow control structures. With this method, we cannot only improve workflow throughput but also complement dynamic deadline allocation methods, while giving less overhead to workflow management systems. In addition, we have discussed how to improve the workflow throughput using the concept of the critical path.

For further work, the escalation cost model appropriate to workflow environments needs to be defined because it directly affects the throughput of time-constrained workflows. We are also studying new dynamic deadline management technologies which can accord with the static activity deadline allocation method proposed in this paper.

Acknowledgments

This work was supported by grant No. 1999-1-303-007-3 from the interdisciplinary Research program of the KOSEF.

References

1. C. Mohan, Recent trends in workflow management products, standards, and research, *Proc. NATO Adv. Study Inst. Workflow Management Syst. Interoperability*, 1997.
2. P. Lawrence, *Workflow Handbook 1997* (John Wiley & Sons Ltd, 1997).
3. F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques* (Prentice Hall, NJ, 1999).
4. P. Heinl, Exceptions during workflow execution, *Proc. Sixth Int. Conf. Extending Database Tech.*, 1998.
5. C. Hagen and G. Alonso, Flexible exception handling in the Opera process support system, *Proc. 18th IEEE Int. Conf. Distributed Comput. Syst.*, 1998.
6. E. Panagos and M. Rabinovich, Reducing escalation-related costs in WFMSs, *Proc. NATO Adv. Study Inst. Workflow Management Syst. Interoperability* (1997) 106–128.
7. E. Panagos and M. Rabinovich, Predictive workflow management, *3th Int. Workshop NGITS*, 1997.
8. M. Xiong, K. Ramamritham, J. Haritsa and J. A. Stankovic, MIRROR: A state-conscious concurrency control protocol for replicated real-time databases, *Proc. Int. Conf. Adv. Issues E-Commerce Web-Based Inf. Syst.* (1999) 20–29.
9. J. Eder, E. Panagos and M. Rabinovich, Time constraints in workflow systems, *Conf. Adv. Inf. Syst. Eng.* (1999) 286–300.
10. G. Alonso, G. Roger, M. Kamath, D. Agrawal, A. E. Abbadi and C. Mohan, Exotica/FMDC: A workflow management system for mobile and disconnected clients, *J. Distributed Parallel Databases* (1996) 229–247.
11. T. Bauer and P. Dadam, A distributed execution environment for large-scale workflow management systems with subnets and server migration, *Proc. 2nd IFCIS Conf. CoopIS* (1997) 99–108.
12. M. Marazakis and C. Mikolaou, Towards adaptive scheduling of tasks in transactional workflows, Technical Reports, University of Crete, 1995.
13. D. Georgakopoulos, M. Hornick and A. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure, *Distributed Parallel Databases* (1995) 119–153.
14. M. A. Vouk, D. L. Bitzer and R. L. Klevans, Workflow and end-user quality of service issues in web-based education, *IEEE Trans. Knowledge Data Eng.* **14** (1999) 673–687.
15. S. K. Oh, J. H. Son, Y. J. Lee and M. H. Kim, An efficient method for allocating workflow tasks to improve the performance of distributed workflows, *Int. Conf. Computer Sci. Inf.*, 2000.
16. B. Kao and H. Garcia-Molina, Deadline assignment in a distributed soft real-time system, *Proc. 13th Int. Conf. Distributed Comput. Syst.* (1993) 428–437.
17. S. H. Son, R. C. Beckinger and D. A. Baker, DRDB: Towards distributed real-time database services for time-critical active applications, *J. Syst. Software* **42** (1998) 193–204.
18. J. H. Son and M. H. Kim, Improving the performance of time-constrained workflow processing, *J. Syst. Software*, 2000 (to be published).
19. L. Kleinrock, *Queueing systems: Computer applications* (John Wiley & Sons, 1974).
20. R. W. Wolff, *Stochastic Modeling and the Theory of Queues* (Prentice Hall, 1989).
21. R. L. Disney, Queueing networks, *Am. Math. Soc. Proc. Symp. Appl. Math.*, 1981.
22. H. A. Taha, *Operations Research* (Macmillan Publishing Company, 1992).