A stochastic model for workflow QoS evaluation[1]

Yunni Xia, H.P. Wang*, Y. Huang and L. Yuan
School of Electronic Engineering and Computer Science, Peking University, Beijing, China, 100871

Abstract. Quality (QoS) prediction is one of the most important research topics of workflow. In this paper, we propose a stochastic model to evaluate QoS (make-span, reliability and cost) of workflow systems based on QWF-net, which extends traditional WF-net by associating tasks with firing-rate, failure-rate and cost-coefficient. Through a case study, we show that our framework is capable of modeling real-world workflow-based application. Also, Monte-carlo simulation in the case study indicates our analytical methods are consistent with simulation. We also present a sensitivity analysis technique to identify QoS bottleneck.The paper concludes with a comparison between our approach and related work.

Keywords: Workflow, QoS, homogeneous continuous-time markovian process, monte-carlo simulation, sensitivity analysis

Notation

T	Set of tasks in QWF-net	Q	Infinitesimal generator
t_i	The $i^{\rm th}$ task	$U(t)$	State-space
P	Set of places in QWF-net	S_i	The $i^{\rm th}$ state in state-space
p_i	The $i^{\rm th}$ place	T_i	Time-to-termination of S_i
$\lambda(t_i)$	Firing-rate of t_i	MS	Make-span of QWF-net
$\mu(t_i)$	Failure-rate of task t_i	R_i	Reliability of t_i
$lo(t_i)$	Skipping probability of task t_i	RS_i	Reliability of state S_i
$se(t_i)$	Selection probability of task t_i	*Reliability*	Reliability of QWF-net
$\theta(t_i)$	Cost-coefficient of task t_i	CO_i	Cost of executing t_i
C	Cost of QWF-net		

1. Introduction

With the advent and evolution of global scale economies, organizations need to be more competitive, efficient and flexible. In the past decade, workflow techniques [22] have been widely used to address these needs. Workflow aims to help business goals to be achieved with high efficiency by means of sequencing work activities and invoking appropriate human and/or information resources associated with these activities.

The application of workflow techniques requires QoS management. Appropriate control of QoS leads to high efficiency of services and high quality of products, thereby fulfilling customer expectations and achieving customer satisfaction. According to [6], being able to evaluate and manage QoS of workflow has four distinct advantages.

[1] This research is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312004.

*Corresponding author. Tel.: +8610 62765818; E-mail: whpxhy@pku.edu.cn.

ISSN 1058-9244/06/$17.00

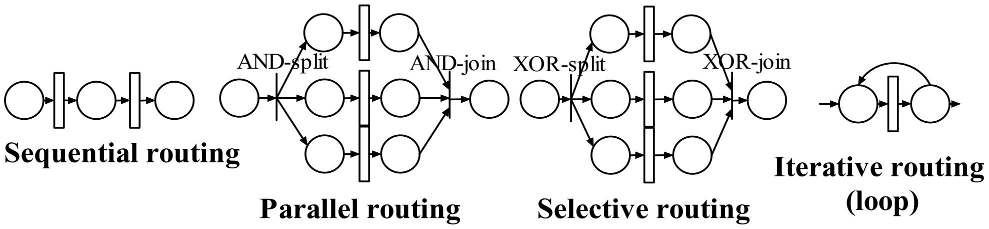


Fig. 1. Routing patterns.

- QoS-based design: it allows organizations to translate their vision into their business processes more efficiently, since workflow can be designed according to QoS metrics. For e-commerce processes it is important to know the QoS an application will exhibit before making the service available to its customers.
- QoS-based selection and execution: it allows for the selection and execution of workflows based on their QoS, to better fulfill customer expectations. As workflow systems carry out more complex and mission-critical applications, QoS analysis serves to ensure that each application meets user requirements
- QoS monitoring: it makes possible the monitoring of workflows based on QoS. Workflows must be rigorously and constantly monitored throughout their life cycles to assure compliance both with initial QoS requirements and targeted objectives. QoS monitoring allows adaptation strategies to be triggered when undesired metrics are identified or when threshold values are reached
- QoS adaptations: to achieve higher performance and reduce cost, it is necessary to expect to adapt, replan, and reschedule workflow system. When adaptation is necessary, a set of potential alternatives is generated, with the objective of changing a workflow as its QoS continues to meet initial requirements. Therefore, through QoS evaluation techniques,workflow designers and managers can study how system adaptations influence performance and decide whether QoS requirements remain satisfied

Among many research topics of workflow, performance/QoS analysis is yet to be given the importance it deserves. Techniques and models for workflow QoS evaluation are still limited. Existing models and approaches fall into two categories, namely simulative and analytical. Among analytical methods [5], derives an analytical model from historical logs, models of [6,9,24] use reduction (simplification) techniques to simplify complex routing constructs into performance-equivalent tasks [11,17], derive analytical models from basic compositional patterns (sequence, parallelism, choice and loop) [10], proposes a performance equivalent analysis technique [5,12,13], model the control flow of workflow systems as continuous Markovian chains, [4] uses a decomposition technique to find some performance bounds of WF-nets [1], introduces a state-based performance evaluation technique for workflow based on stochastic Petri-net.

This paper introduces an analytical approach to address the need for QoS evaluation. This approach is based on QWF-net (WF-net for QoS evaluation) model, which is an extension of traditional WF-net by associating tasks with firing-rate, failure-rate and cost coefficient. By mapping the execution process of QWF-net into a continuous Markovian process, analytical methods to evaluate make-span(expectation and standard-deviation calculated), cost(expectation and standard-deviation calculated) and reliability is developed. The case-study shows that our approach is capable of modeling real-world workflow applications. The Monte-carlo simulation in the case study indicates our analytical methods are consistent with simulation. For the purpose of finding QoS bottlenecks, a sensitivity analysis technique is also proposed based on models above. The technique is capable of determining which task influence the system QoS most and therefore deserves optimization most. The idea of sensitivity analysis is inspired by [7,21].

2. QWF-NET for QoS evaluation

The Workflow net (WF-net) proposed by van der Aalst [22] is a high level Petri Nets with two special places i and o, which indicate the beginning and the end of the modeled process. Every transition is on a path, and a fork and a join transition bound each path. A fork is a transition with more than one output places and a join is a transition

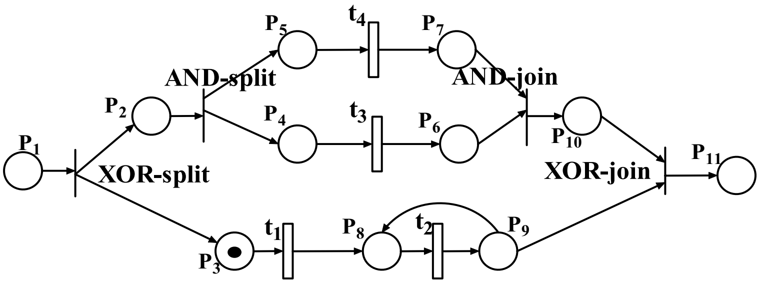


Fig. 2. A WF-net sample.

with more than one input places. WF-net incorporates four routing patterns namely sequence, parallelism, selection and loop (illustrated by Fig. 1.). Figure 2 illustrates a WF-net sample.

Definition 1. (**WF-net**) A Petri Net $N_1 = (P, T, F)$ is a WF-net (Workflow net) if and only if:

- There is one source place $i \in P$ such that $\cdot i = \varnothing$.
- There is one sink place $o \in P$ such that $i\cdot = \varnothing$.
- Every node $x \in P \cup T$ is on a path from i to o.

WF-net does not care the concept of QoS, but sometimes we need to consider QoS aspect in real-world applications. For example, we want to know the time that workflow instance takes to travel from beginning to end (make-span)in a workflow net so that we can decide whether the arrangement of the workflow system meets our time requirement. So introducing QoS concept into WF-net is necessary.

For QoS evaluation, some quantitative information must be obtained, such as the execution-duration/firing-delay of each task, the TTF (time-to-failure) of each task and the probability that each branch on XOR-split (selective routing) is selected. This paper assumes every task has independent random firing-delay/TTF and each associated with a firing-rate/failure-rate. Formally, we extend WF-net to QWF-net (meaning WF-net for QoS analysis) by

Definition 2. (**QWF-net**) $N_2 = (P, T, \mathit{Task}, \lambda, \mu, \theta, \mathit{se,lo})$ is a QWF-net if and only if:

- N_2 is structurally a WF-net.
- SPLIT/JOIN transitions (transitions illustrated by black thin bars in Fig. 2) fire immediately and have firing-delay of 0.
- SPLIT/JOIN transitions never fail.
- The set $\mathit{Task} \subseteq T$ denotes the set of transitions excluding SPLIT/JOIN transitions, as illustrated by white bars in Fig. 2. The i^{th} task is recorded as t_i.
- Firing-rate denotes the probability that task finishes execution at time $t + \Delta t$ (where Δt is an infinitesimal period) if it is still busy at time t. A function $\boldsymbol{\lambda} : \boldsymbol{Task} \longrightarrow \boldsymbol{Real}$ is used to identify the **firing-rate** of each task. In practice, firing-rate is quantitatively measured by the reciprocal of mean firing-delay

$$\lambda(t_i) = \lim_{\delta \to 0} \frac{P\{t_i \textit{ idle at } t + \delta | \textit{active at } t\}}{\delta} = \frac{1}{\textit{Mean firing delay of } t_i} \tag{1}$$

In practice, this estimate can be obtained as the mean firing-delay/execution-duration of the task in its history

- Failure-rate denotes the probability that task fails at time $t + \Delta t$ (where Δt is an infinitesimal period) if its is still correct at time t. A function $\boldsymbol{\mu} : \boldsymbol{T} \longrightarrow \boldsymbol{Real}$ is used to identify the failure rate of each task. In practice, the failure-rate of task t_i is measured by the reciprocal of mean-TTF

$$\mu(t_i) = \lim_{\delta \to 0} \frac{P\{t_i \textit{ down at } t + \delta | \textit{correct at } t\}}{\delta} = \frac{1}{\textit{Mean TTF of } t_i} \tag{2}$$

In practice, this estimate can be obtained as the mean TTF of the task in its history

- A function $\boldsymbol{\theta} : \boldsymbol{Task} \longrightarrow \boldsymbol{Real}$ is used to identify the **cost-coefficient** of each task. The cost of executing a task is the product of its firing-delay and its cost coefficient

- The control flow randomly chooses its path along **XOR-split**. For generality, this paper uses a function $\boldsymbol{se} : \boldsymbol{Task} \longrightarrow \boldsymbol{Real}$ to denote the probability that each task is selected when its corresponding XOR-split is activated. Note that, if a task is not on any XOR-split, its choice probability equals 1, otherwise smaller than 1. The choice probabilities of tasks on the same XOR-split sum up to 1.
- The control flow skips **loop** when its current iteration finishes according to some probability. For generality, this paper uses a function $\boldsymbol{lo} : \boldsymbol{Task} \longrightarrow \boldsymbol{Real}$ to denote the skipping probability of each task. Note that, if a task is not on any loop, its skipping probability equals 1, otherwise smaller than 1.

It easily follows that QWF-net is identical with WF-net in construction aspect. Also, structural properties of WF-net follow in QWF-net: there should be no dead tasks; the procedure should terminate eventually; at the moment the procedure terminates there should be one token in sink place o and all the other places are empty; the definition of reachable markings and its corresponding calculation methods for WF-net can also be applied to QWF-net. Note that, analytical methods based on QWF-net rely on the mapping from execution process of QWF-net into a continuous Markovian process. Such a mapping is under the assumption that transition between Markovian states depends on the current state only. Although one could argue that in reality such assumption may not always be held, the research by [14,15] conclude that many experiments of large-scale software closely converged to the Markovian process results after a long duration. Therefore, Markovian models is well competent as an analytical framework for QoS evaluation of workflow systems.

3. QoS evaluation based on QWF-NET

In this sections, we introduce analytical models to evaluate the make-span, reliability and cost of QWF-net.

Let D_i denote the firing-delay of task t_i and M_i denote the number of loop iterations. M_i is geometrically distributed with parameter $lo(t_i)$. Then, the cumulative distribution function (CDF) of D_i is given as

$$\begin{aligned} F(y) = P\{D_i \leqslant y\} &= \sum_{K=1}^{\infty} P\{M_i = K\} P\{M_i \times X_i \leqslant y | M_i = K\} \\ &= \sum_{K=1}^{\infty} lo(t_i)(1 - lo(t_i))^{K-1} E_K(y) \end{aligned} \tag{3}$$

Where X_i denotes duration of one single iteration of t_i (which is exponential due to constant firing-rate $\lambda(t_i)$), $E_K(y)$ denotes the CDF of **K-phase Erlang distribution**. Then, the probability density function (PDF) of D_i is given as

$$\begin{aligned} f(y) = F'(y) &= \sum_{K=1}^{\infty} lo(t_i)(1 - lo(t_i))^{K-1} \frac{\lambda(t_i)(y\lambda(t_i))^{K-1}}{(K-1)!} e^{-\lambda(t_i)y} \\ &= \lambda(t_i) lo(t_i) e^{-y\lambda(t_i)} \sum_{K=1}^{\infty} \frac{((1 - lo(t_i))y\lambda(t_i))^{K-1}}{(K-1)!} \\ &= \lambda(t_i) lo(t_i) e^{-y\lambda(t_i)} \times e^{(1-lo(t_i))\lambda(t_i)y} \\ &= \lambda(t_i) lo(t_i) e^{-\lambda(t_i) lo(t_i) y} \end{aligned} \tag{4}$$

where $\frac{\lambda(t_i)(y\lambda(t_i))^{K-1}}{(K-1)!} e^{-\lambda(t_i)y}$ is the PDF of the K-phase Erlang distribution.

Equation (4) indicates that D_i follows exponential distribution with parameter $\lambda(t_i) lo(t_i)$.

Let $U(t)$ denote the set of active tasks in QWF-net at time t (execution begins at time 0), then its state-space (denoted by S) is obtained through mapping each reachable marking into a corresponding set of active tasks. For any reachable marking where no SPLIT/JOIN transitions are activated, there exists a state which records all active tasks in this marking. For instance, the marking illustrated in Fig. 2, $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$, where only place p_3 contains a token is mapped into a $U(t)$ state $\{t_1\}$. Note that the marking where only sink place contains a token is

Table 1
Reachable markings and their corresponding states in $X(t)$

Reachable Markings	Marking vector	Corresponding State	Active tasks
M_0	[10000000000]		
M_1	[01000000000]		
M_2	[00100000000]	S_1(Initial)	$\{t_1\}$
M_3	[00011000000]	S_2(Initial)	$\{t_3, t_4\}$
M_4	[00000001000]	S_3	$\{t_2\}$
M_5	[00001100000]	S_5	$\{t_4\}$
M_6	[00010010000]	S_4	$\{t_3\}$
M_7	[00000000010]	S_6(Absorbing)	$\varnothing$
M_8	[00000000100]	S_6	$\varnothing$
M_9	[00000000001]	S_6	$\varnothing$

mapped into an **absorbing state** which records no task is active, meaning the termination of control flow. The state space of $U(t)$ of Fig. 2 are illustrated in Table 1. For any reachable marking, the i^{th} entry in the marking-vector indicates place P_i is empty if the entry is 0 or full otherwise. Note that, there exist more than one **initial-state** since $XOR - SPLIT_1$ may generate token into place either p_2 or p_3. S_{11} is the absorbing state.

Since D_i for each task is exponential according to Eq. (4), $U(t)$ is a homogeneous continuous Markovian process. The infinitesimal generator matrix Q of $U(t)$ is given as

$$q_{i,j} = \begin{cases} lo(t_l) \times \lambda(t_l) \times \prod_{t_m \in NEW(i,j)} se(t_m) & \text{if } S_i \xrightarrow{t_1} S_j \\ -\sum_{1 \leqslant r \leqslant |S|, r \neq i} q_{i,r} & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{5}$$

where $lo(t_l) \times \lambda(t_l)$ is the parameter of exponential D_l, $|S|$ denotes the number of states in the state space, and $q_{i,j}$ denotes the transition rate from state S_i to S_j. Relation $S_i \xrightarrow{t_l} S_j$ means that S_j is the resulting state of S_i if the active task t_l in S_i finishes execution and becomes idle. Note that, there may exist more than one resulting states of S_i when t_l becomes idle because choice (XOR-split) may be activated. Those resulting states are viewed as different **types** in the Markovian chain according to the **phase-type** property [20]. $\prod_{t_m \in NEW(i,j)} se(t_m)$ denotes the occurrence probability of S_j among all types, where **NEW(i,j)** denotes the set of newly-emerging active tasks in the transition from state S_i to S_j.

3.1. Evaluating make-span

Time is a common and universal measure of performance. The philosophy behind a time-based strategy usually demands that businesses deliver the most value as rapidly as possible. Shorter workflow execution time allows for a faster production of new products, thus providing a competitive advantage.

The first measure of time is task firing-delay. Task firing-delay corresponds to the time an instance takes to be processed by a task. According to [6], the time can be broken down into two major pieces: wait time (WT) and process time (PT). Wait time refers to the non-value-added time needed in order for an instance to be processed by a task. This includes, for example, the instance queuing wait (QW) and the setup wait (SW) of the task. While, those two metrics are part of the task operation, they do not add any value to it. Queuing wait is the time instances spend waiting in a task-list, before the instance is selected for processing. Setup wait is the time an instance spends waiting for the task to be set up. Setup activities may correspond to the warming process carried out by a machine before executing any operation, or to the execution of self-checking procedures. Process time is the time a workflow instance takes at a task while being processed; in other words, it corresponds to the time a task needs to process an instance. In an general way, we use firing-rate to stochastically depicts the random firing-delay of each task in QWF-net.

In this paper, make-span (denoted by random variable *MS* is defined as the time that workflow instance takes to travel from source place i to sink place o. Shorter make-span means faster completion of workflow applications and higher efficiency.

Let W_i denote the time for state S_i to reach the absorbing state (time-to-termination). The moments of W_i are given by the following theorem

Theorem 1. [Moments of time-to-termination]

$$E(W_i^n) = \frac{nE(W_i^{n-1}) + \sum_{1\leqslant k\leqslant |S|, k\neq i} q_{i,k}E(W_k^n)}{Z_i} \tag{6}$$

where $E(W_i^0) = 1$, and $E(W_i^n) = 0$ if S_i is the absorbing-state and $n \geqslant 1$. Z_i is given by $Z_i = \sum_{1\leqslant j\leqslant |S|, j\neq i} q_{i,j}$
proof: Let V_i denote the elapsing time of state S_i and $O_i = W_i - V_i$ representing the time to termination just after the Markovian process $U(t)$ leaves state S_i. O_i are related to those immediately succeeding states of S_i. Its moments are given by

$$E[O_i^n] = \sum_{1\leqslant k\leqslant |S|, k\neq i} \frac{q_{i,k}}{Z_i} E(W_k^n) \tag{7}$$

where $\frac{q_{i,k}}{Z_i}$ is the probability that state S_k immediately succeeds S_i. $E[O_i^n]$ is the weighted (by occurrence probability) moments of S_i's immediately-succeeding states.

V_i and O_i are independent of each other. Since V_i follows exponential distribution with parameter Z_i, we have its moments as

$$E(V_i^n) = \frac{n!}{Z_i^n} \tag{8}$$

Then, $E(W_i^n)$ is given by

$$\begin{aligned} E(W_i^n) &= E[(V_i + O_i)^n] \\ &= E(O_i^n) + E(V_i^n) + E\left[\sum_{j=1}^{n-1} C_n^j O_i^j V_i^{n-j}\right] \end{aligned} \tag{9}$$

Since O_i^j and V_i^{n-j} are independent of each other, we have

$$E(O_i^j V_i^{n-j}) = E(O_i^j)E(V_i^{n-j}) \tag{10}$$

Consequently

$$\begin{aligned} E\left[\sum_{j=1}^{n-1} C_n^j O_i^j V_i^{n-j}\right] &= \sum_{j=1}^{n-1} C_n^j E[O_i^j]E[V_i^{n-j}] = \sum_{j=1}^{n-1} C_n^j E[O_i^j]\frac{(n-j)!}{Z_i^{n-j}} \\ &= \sum_{j=1}^{n-1} C_n^j E[O_i^j]E[V_i^{n-1-j}]\frac{n-j}{Z_i} = E\left[\sum_{j=1}^{n-1} C_n^j O_i^j V_i^{n-1-j}\frac{n-j}{Z_i}\right] \\ &= \frac{n}{Z_i}E\left[\sum_{j=1}^{n-1} C_{n-1}^j O_i^j V_i^{n-1-j}\right] \end{aligned} \tag{11}$$

Therefore, we have

$$\begin{aligned} E(W_i^n) &= E(O_i^n) + E(V_i^n) + \frac{n}{Z_i}E\left[\sum_{j=1}^{n-1} C_{n-1}^j O_i^j V_i^{n-1-j}\right] \\ &= E(O_i^n) + \frac{n}{Z_i}\frac{(n-1)!}{Z_i^{n-1}} + \frac{n}{Z_i}E\left[\sum_{j=1}^{n-1} C_{n-1}^j O_i^j V_i^{n-1-j}\right] \\ &= E(O_i^n) + \frac{n}{Z_i}E\left[V_i^{n-1} + \sum_{j=1}^{n-1} C_{n-1}^j O_i^j V_i^{n-1-j}\right] \end{aligned} \tag{12}$$

$$= E(O_i^n) + \frac{n}{Z_i} E\left[(O_i + V_i)^{n-1}\right] = E(O_i^n) + \frac{n}{Z_i} E[W_i^{n-1}]$$

$$= \frac{nE[W_i^{n-1}] + \sum_{1 \leqslant k \leqslant |S|, k \neq i} q_{ik} E(W_k^n)}{Z_i}$$

Therefore, the theorem follows. □

Then the moments of make-span *MS* are obtained as the weighted moments of all initial-states' time-to-termination

$$E(MS^n) = \sum_{S_i \in Init} E(W_i^n) \times \prod_{t_j \in AT_i} se(t_j) \tag{13}$$

Where *Init* denotes the set of initial states and AT_i denotes the set of active tasks in state S_i. $\prod_{t_j \in AT_i} se(t_j)$ is the occurrence probability of initial state S_i.

Moreover, the standard deviation of MS is obtained as

$$\sigma(MS) = \sqrt{E(MS^2) - E^2(MS)} \tag{14}$$

3.2. Evaluating reliability

To model the reliability dimension of workflow system, this paper applies software reliability theories to the QWF-net model. The first step is to model the reliability of individual tasks.

Most software reliability researches assume that information about task's failure behaviors is available and its failure-rate is known, that is, ignore the issue of how they can be determined. Assessing the reliability of individual software modules clearly depends on the factors such as whether or not task code is available, how well the task has been tested, and whether it is a reused task or a new task. **Reliability growth model** has been widely accepted as a reasonable solution for identifying failure-rate of software modules. For example [8], applied the model based on non-homogeneous Poisson process, the hyper-exponential model, in order to estimate the stationary failure rates of software modules. [19] used the enhanced NHPP model, proposing a method for determining task's time-dependent failure intensity based on block coverage measurement during the testing. [23] identified guidelines for estimating failure-rate of the newly developed software modules based on the EET model whose parameters are related to the task's static and dynamic properties and the usage of each task.

This paper also depicts the reliability of workflow task through its failure-rate. Our approach of reliability estimation differs from many existing methods (for instance [6]) in that, it considers reliability of task to be dependent on many factors (firing-rate, failure-rate) rather than directly assign a independent reliability estimate to each task, as explained by Eq. (16).

Since each task in QWF-net has constant failure-rate, we have

$$Prob\{TTF_{t_i} > t\} = e^{-\mu(t_i)t} \tag{15}$$

We use R_i to denote the reliability estimate of task t_i. R_i is obtained through integrating the probability that t_i remain correct at time t (or the probability that TTF_i is larger than t) multiplied by the probability-density-function (PDF) of the firing-delay of D_i over the time interval from 0 to ∞. Therefore

$$\begin{aligned} R_i &= \int_0^\infty \lambda(t_i) lo(t_i) e^{-\lambda(t_i) lo(t_i) t} \times Prob\{TTF_{t_i} > t\} dt \\ &= \int_0^\infty \lambda(t_i) lo(t_i) e^{-\lambda(t_i) lo(t_i) t} \times e^{-\mu(t_i)t} dt \\ &= \frac{\lambda(t_i) lo(t_i) \int_0^\infty (\lambda(t_i) lo(t_i) + \mu(t_i)) e^{-(\lambda(t_i) lo(t_i) + \mu(t_i))t} dt}{\lambda(t_i) lo(t_i) + \mu(t_i)} \\ &= \frac{\lambda(t_i) lo(t_i)}{\lambda(t_i) lo(t_i) + \mu(t_i)} \end{aligned} \tag{16}$$

where $\lambda(t_i)lo(t_i)e^{-\lambda(t_i)lo(t_i)t}$ is the probability-density-function of the firing-delay D_i and $e^{-\mu(t_i)t}$ is the probability that TTF_i is greater than t.

The reliability of QWF-net is obtained as the weighted reliability of its initial states

$$Reliability = \sum_{S_i \in Init} \left(RS_i \times \prod_{t_j \in AT_i} se(t_j) \right) \tag{17}$$

where RS_i denotes the reliability of state S_i, meaning the probability QWF-net keeps correct from state S_i to the absorbing state. RS_i is given by

$$RS_i = \begin{cases} 1 & \text{Absorbing-state} \\ \sum_{Every\ S_j\ where\ S_i \xrightarrow{t_l} S_j} \frac{q_{i,j}}{Z_i} R_l \times RS_j & \text{Else} \end{cases} \tag{18}$$

where $\frac{q_{i,j}}{Z_i}$ denotes the probability that state S_j immediately succeeds S_i.

3.3. Evaluating cost

During workflow design, both prior to workflow instantiation and during workflow execution, it is necessary to estimate the cost of the execution in order to guarantee that financial plans are followed.

The cost of QWF-net is the cost of running all scheduled tasks in QWF-net. The total cost is dependent both on cost of each task and the structure of QWF-net. [6] gives a detailed discussion of workflow's execution cost. However, it assumes that the task cost is constant and independent of its firing-delay. In this paper however, we assume task cost is the product of its firing-delay and cost-coefficient. Let CO_i denote task cost of t_i, we have its moments as

$$E(CO_i^n) = \theta(t_i)^n \times E(D_i^n) = \theta(t_i)^n \times \frac{n!}{\lambda(t_i)^n} \tag{19}$$

where $\frac{n!}{\lambda(t_i)^n}$ denotes the moments of firing-delay D_i.

Let PE_i denote the probability that task t_i is executed. To calculate PE_i, we first define the occurrence probability of state S_i as OC_i. OC_i is given by

$$OC_i = \begin{cases} \prod_{t_j \in AT_i} se(t_j) & \text{Initial-state} \\ \sum_{ALL\ S_j\ satisfying\ S_j \xrightarrow{t_l} S_i} OC_j \times \frac{q_{j,i}}{Z_j} & \text{Else} \end{cases} \tag{20}$$

where AT_i denotes the set of active tasks in state S_i and Z_j is given earlier by Eq. (6).

Then, PE_i is given by

$$PE_i = \sum_{ALL\ S_j\ satisfying\ S_j \xrightarrow{t_i} S_k} OC_j \times \frac{q_{j,k}}{Z_j} \tag{21}$$

Then, the cost of QWF-net, C, is defined as the sum of each task's cost multiplied by its probability of being executed. Its expectation is

$$E(C) = \sum_{t_i \in\ QWF\text{-}net} E(CO_i) \times PE_i \tag{22}$$

Also, the second-order moment of C is

$$\begin{aligned} E(C^2) &= E\left(\left(\sum_{t_i \in N_2} PE_i CO_i \right)^2 \right) \\ &= \sum_{t_i \in N_2} PE_i \times ((PE_i)E(CO_i^2) + \sum_{t_j \in N_2 \wedge t_i \neq t_j} E(CO_i)PE_j E(CO_j)) \end{aligned} \tag{23}$$

The standard deviation of C is given by

$$\sigma(C) = \sqrt{E(C^2) - E^2(C)} \tag{24}$$

Table 2
Tasks in the case study

T	λ	μ	θ	se	lo	T	λ	μ	θ	se	lo
t_1	0.3	0.0026	1.1	1	1	t_7	0.30	0.0013	0.8	0.2	1
t_2	0.2	0.0028	1.2	1	1	t_8	0.45	0.0014	0.6	0.1	1
t_3	0.4	0.0053	2.1	1	0.33	t_9	0.30	0.0064	0.4	1	1
t_4	0.65	0.0041	0.65	1	1	t_{10}	0.45	0.0054	1.1	1	1
t_5	0.6	0.0051	1.0	1	1	t_{11}	0.80	0.0023	0.4	1	1
t_6	0.2	0.0037	1.4	0.7	1	t_{12}	0.25	0.0058	0.45	1	1

4. Case study and Monte-carlo simulation

Often, an analytical approach is preferred over simulation. However, the complexity of a real software system can be such that a simulation approach is the only feasible means of analysis. Given a specific process model, there are several aspects that determine whether an analytical approach is feasible at all and-if so- preferable over simulation. For example, if random duration is too complex (for example general distribution) or the state space is too large, it will be extremely difficult to obtain an analytical solution. Simulation is then the only possible alternative to obtain quantitative solution.

Moreover, even if analytical models are tractable, simulation is still useful in that modelers can figure out whether analytical models are correct and accurate by comparing simulative and analytical solutions.

This section applies our analytical framework to some examples and studies performance in a simulative manner. Monte-carlo simulation is a flexible performance prediction tool used widely in science and engineering [2,3, 16]. Its flexibility stems from the fact that it consists of a computer program that behaves like the system under study. The stochastic behaviors and events of target system are modeled using pseudo-random number generators. The execution of a computer simulation is comparable to conducting an *in-vitro* experiment on the target system. Simulation outputs are treated as random observations (samples). The idea of monte-carlo simulation in this section is also inspired by the approach of discrete-event simulation [18] to analyze component-based software system. This approach relies on random generation of faults in components using a programmatic procedure which returns the inter-failure arrival time of a given component. The total number of failures is calculated for the application under simulation, and its reliability is estimated. This approach assumes the existence of a control flow graph of a program. The simulation approach assumes failure and repair rates for components, and uses them to generate failures in executing the application. It also assumes constant execution time per component interaction, and ignores failures in component interfaces and links (transition reliabilities).

The cases studied are given by Fig. 3. Firing-rates, failure-rates, cost-coefficients of involved tasks are listed in Table 2. Cases c_{1-4} are four simple cases dealing with sequential, parallel, selective and iterative routing modes, respectively.

c_7 is a complex case featured by all the four routing styles. It models a booking-order processing workflow application, including several tasks responsible for different functions. The application acquires booker's information, checks previous booking records in the customer's account, then processes in parallel the insurance-related services, booking task and secondary service. The booking task t_3 is executed iteratively because customer may have more than one booking orders. The insurance service ($t_{4,6,7,8,9}$) first gets the insurance account information of the booker and then provides three optional insurance services and runs the banking procedure. Note that, tasks $t_{6,7,8}$ are on a XOR-split meaning customers can choose from three kinds of optional services. After tasks above accomplish their execution, updating and maintenance tasks will be executed to make accounts correct and up-to-date.

The Monte-carlo simulation procedure (pseudocode given in Fig. 4) conducts h ($h = 50000$ in our simulation program) experiments of the execution process of target QWF-net. At each experiment, the procedure randomly selects its paths along XOR-splits according to choice probability given in Table 2. Then, it uses random variable generator to generate the number of iterations of each iteratively-executed task. In the following, it uses random variable generator again to generate firing-delay/TTF of each task according to firing-rate/failure-rate given in Table 2, thereby changing stochastic QWF-net into a deterministic one. The make-span of QWF-net at current experiment is then obtained as the longest path from source-place to sink-place of the deterministic QWF-net. The cost of QWF-net is obtained as the sum of all involved tasks' cost. At each experiment, if every task's TTF is greater than its

Table 3
Comparison between analytical and simulative results

	$E(MS)/\overline{MS}$	$\sigma(MS)/S(MS)$	Reliability/Rel	$E(C)/\overline{C}$	$\sigma(C)/S(C)$
c_1	**9.87**/9.88	**6.20**/6.20	**97.16%**/97.22%	**10.67**/10.67	**7.10**/7.10
c_2	**4.22**/4.22	**3.09**/3.10	**96.48%**/96.45%	**4.00**/4.01	**2.36**/2.35
c_3	**4.39**/4.40	**4.60**/4.59	**98.61%**/98.62%	**5.57**/5.59	**6.39**/6.38
c_4	**7.57**/7.57	**7.58**/7.56	**96.14%**/96.08%	**15.91**/15.96	**15.91**/15.88
c_5	**7.55**/7.57	**4.55**/4.59	**95.65%**/95.55%	**7.67**/7.67	**4.36**/4.37
c_6	**12.30**/12.31	**7.77**/7.76	**93.49%**/93.56%	**27.48**/27.51	**18.16**/18.15
c_7	**23.28**/23.27	**9.35**/9.36	**86.11%**/86.15%	**39.89**/39.89	**18.93**/18.94

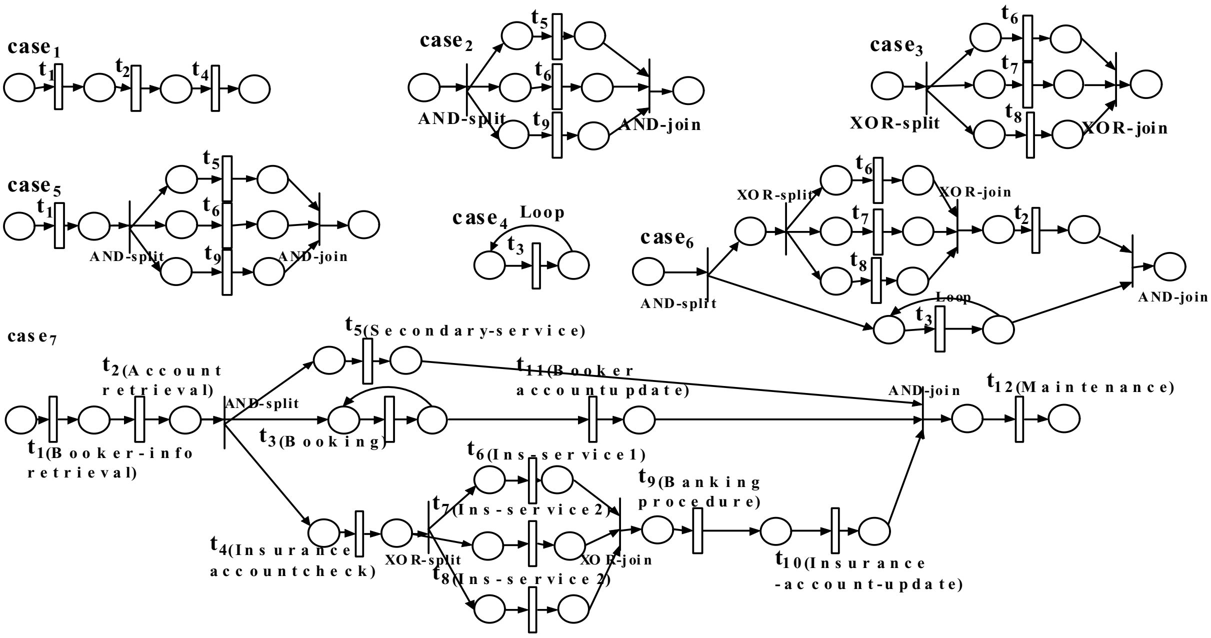


Fig. 3. Cases c_{1-7}.

firing-delay (meaning no failure happens during the execution of this task), a success is recorded. The sample-mean of make-span ($\overline{MS}$), sample-mean of cost ($\overline{C}$), standard-deviation of make-span ($S(MS)$) and standard-deviation of cost ($S(C)$) are then obtained as the mean make-span of all experiments, the mean QWF-net's cost of all experiments, the standard-deviation of all experiments' make-spans and the standard-deviation of all experiments' QWF-net cost, respectively. The simulative estimate of reliability (*Rel*) is the ratio of successes to the number of experiments.

Results obtained by Monte-carlo simulation (illustrated in normal style) are compared with those by our analytical approach (illustrated in bold style) in Table 3. As shown, analytical results is pretty close to simulative results, indicating our analytical approach is consistent with simulation.

5. Sensitivity analysis

Sensitivity analysis is another important aspect in QoS analysis. It is very useful for bottleneck analysis and optimization of the software. During the design stage it is common that the exact values of the input parameters for the model are unknown. Sensitivity analysis can then help in analyzing the influence of the change in input parameters on the performance and reliability metrics.

More over, the QoS of a workflow system can increase in its life cycle if improvements of some tasks are carried out. Therefore, considering such a system we are often interested to know which task is more important than others.

```
algorithm Monte-carlo simulation
Input: QWF-net
Output: Mean/Standard-deviation of make-span/cost, Reliability

• FOR $1 \leq i \leq h$

        Randomly selects its paths along XOR-splits
        Generate the number of iterations of each iteratively-executed task
        Generate firing-delay/TTF of each task
        calculate the system make-span and cost
        If some task has a TTF < firing-delay then record a failure

• END FOR
• $\overline{COST}$ = averaged cost of all experiments
• $S(COST)$ = standard-dev cost of all experiments
• $\overline{MS}$ = averaged make-span of all experiments
• $S(MS)$ = standard-dev make-span of all experiments
• $Rel$ = ratio of successes to $h$
```

Fig. 4. Monte-carlo simulation.

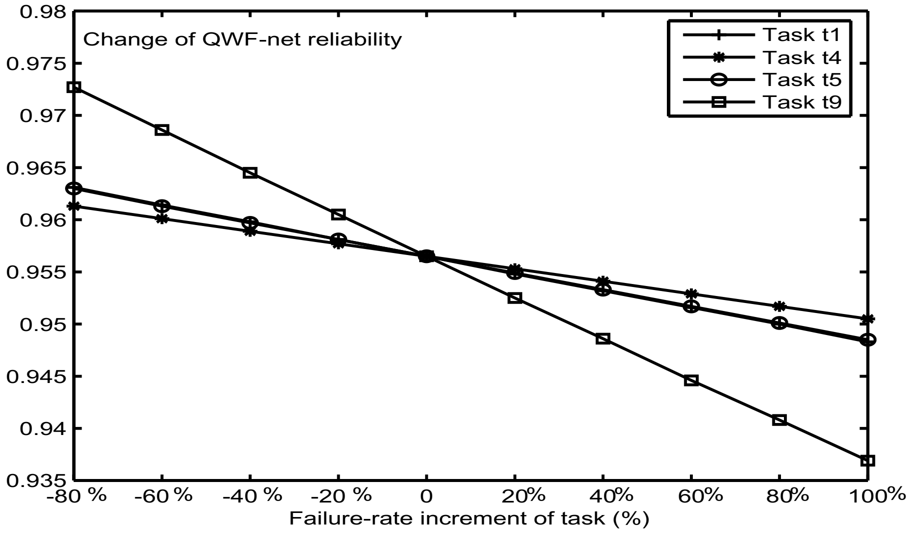


Fig. 5. Sens of reliability.

Thus, the improvement of that critical task will increase the system QoS more than others. Sensitivity analysis gives an approach to analyze the relative importance of consisting tasks in determining which task deserves optimization most.

Based on analytical methods presented in previous sections, this section presents a sensitivity-analysis technique. To decide the sensitivity of system QoS to a specific task, we simply change the task's input QoS parameters while stabilizing other tasks and study whether the total QoS fluctuates dramatically. If strong fluctuation of system QoS occurs, we can conclude the system is very sensitive to the task under study and that task deserves optimization most. The analysis is conducted on case c_5 shown by Fig. 3 and input performance parameters of related tasks are given by by Table 2. To study the sensitivity of reliability to a specific task, we change its failure-rate by between -80%

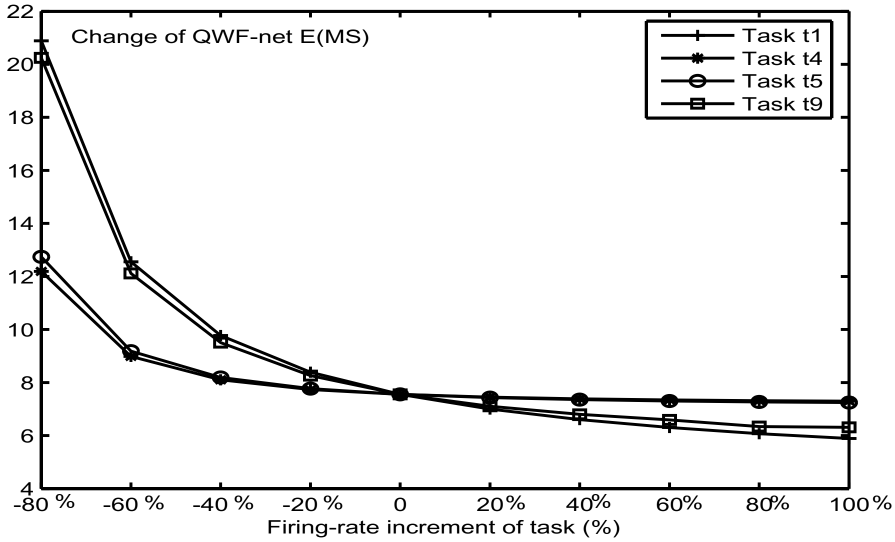


Fig. 6. Sens of $E(MS)$.

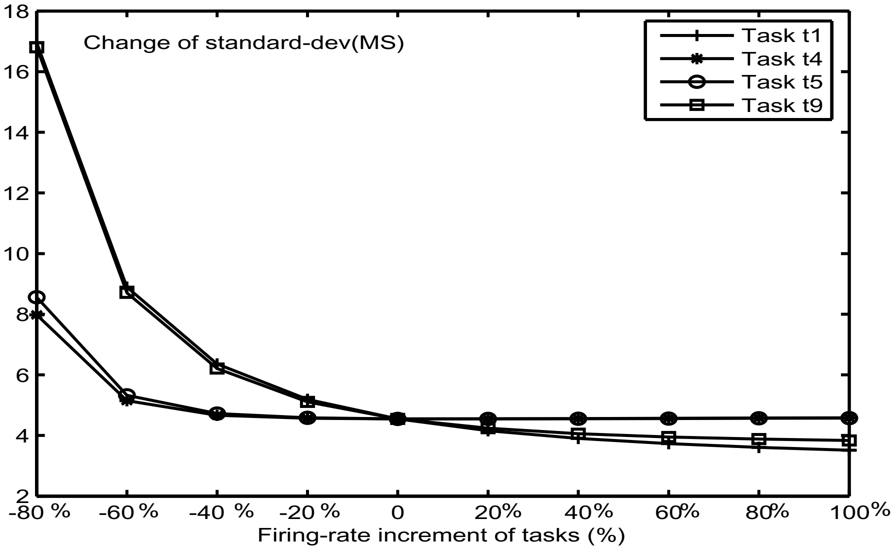


Fig. 7. Sens of $\sigma(MS)$.

and 100% and stabilize failure-rates of other tasks. The sensitivity of $E(MS)/\sigma(MS)/E(C)/\sigma(C)$ is obtained in a similar way through changing execution-rate/cost-coefficient of each task. Figs 5–9 illustrates the fluctuation of system performance after input parameter of every task is changed. According to Figs 5–9, we have that system reliability is most sensitive to task t_9 while make-span and cost are most sensitive to task t_1, meaning that those tasks deserve improvement most and optimizing those tasks makes greatest sense.

Note that, our analysis of sensitivity is different from those differentiation-based techniques. For instance, [7,21] establish symbolic equations relating system performance and input performance parameters. Through differentiating the symbolic equation, they use partial derivatives as the sensitivity estimates of input parameters. However, this approach is feasible only if the symbolic equation is obtainable and tractable. If system performance is determined by too many input parameters or the symbolic equation is given in an iterative manner (just like Eq. (20)), partial derivatives are not likely to obtain easily.

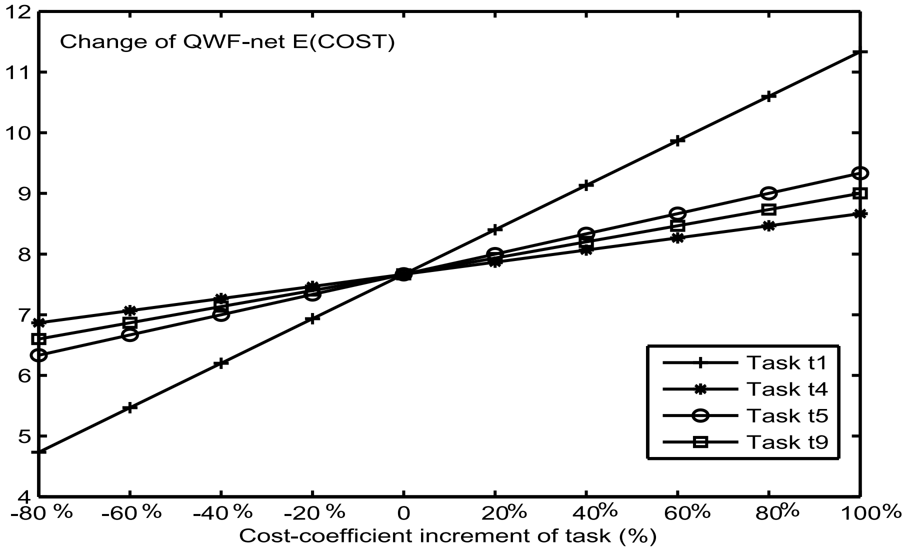


Fig. 8. Sens of $E(C)$.

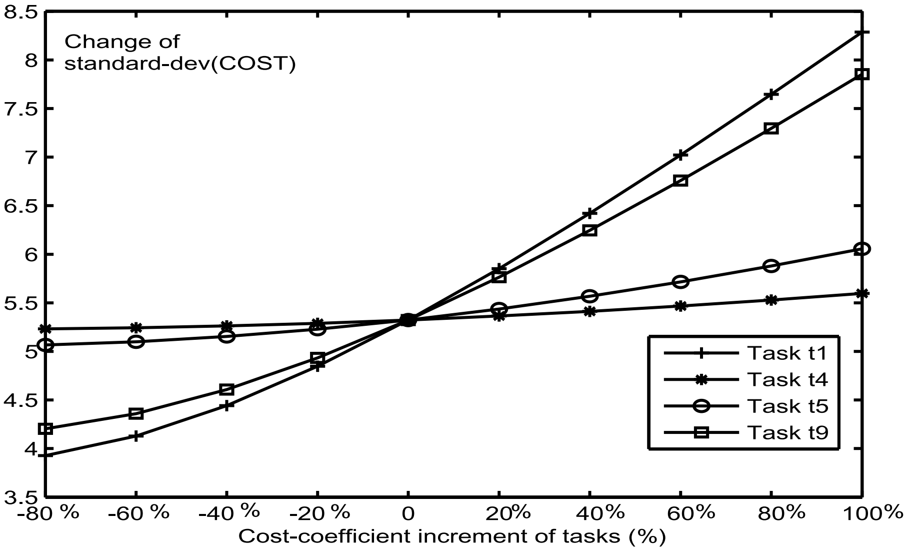


Fig. 9. Sens of $\sigma(C)$.

6. Comparison with related work

The method of [5] derives an analytical model from historical logs. For each situation in the logs, the method generates a corresponding state in the state space. Therefore this method can only be feasible when historical logs are small, otherwise it might cause state explosion.

Research of [6,9,24] uses reduction techniques to simplify sequential, parallel, selective and iterative routing patterns into a single transition with equivalent QoS estimate, however their models are not very realistic since they assume tasks in WF-net have deterministic firing-delay.

Methods of [5,12,13] are similar to our approach in that they all model the control flow as Markov processes. These models assume execution of each task to be each state in the Markov chain and develop methods to evaluate transition-probability (for DTMC) or transition-rate (for CTMC) between states. However these Markov-based methods can not model parallel execution of more than one tasks.

Method proposed by [10] is similar to our model in that they both associate task with firing-rate. It develops a simplification technique to simplify four basic routing patterns into a single task with approximate equivalent performance. This method is pretty simple and intuitive comparing to models based on Markovian processes. However, the assumption that the performance-equivalent task follows exponential distribution is obviously inaccurate. For instance, n tasks with the same constant firing-rate arranged by sequential routing pattern should be simplified into a single equivalent task of n-phase Erlang firing-delay (which is obviously not exponential). Moreover, this contribution only calculates the expectation of system make-span (In contrast, our model also gives its standard deviation).

[1] introduces a performance-evaluation technique through mapping WF-net into stochastic Petri-net. This contribution also assigns each task with a constant firing-rate. However, this research is only focused on stochastic modeling rather than performance evaluation because no methods to evaluate performance metrics are given.

As for sensitivity analysis, our approach differs from [21] and [7] in that our analysis is conducted in a numerical way. [7,21] establishes symbolic equation relating system performance and input performance parameters and uses partial derivatives as the sensitivity estimates of input QoS parameters. However, this approach is feasible only if the symbolic equation is obtainable and tractable. If system QoS is determined by too many input parameters or the symbolic equation is given in an iterative manner (just like Eq. (20)), partial derivatives are not likely to obtain easily.

7. Conclusion and further study

To stochastically model the execution of workflow systems, this paper extends traditional WF-net to QWF-net through associating workflow task with firing-rate, failure-rate and cost-coefficient. Through mapping the execution of QWF-net into a homogeneous CTMC process, we presents analytical methods to evaluate make-span, reliability and cost of QWF-net. This paper also develops a Monte-carlo simulation procedure to calculate the simulative QoS results. The comparison between simulative results and analytical results in the case study indicates that our approach is consistent with simulation. For the purpose of QoS bottle-neck identification, a sensitivity-analysis technique is also introduced.

We are currently making some further study extending this paper

– Introducing hierarchical QWF-net model and corresponding analytical methods for QoS evaluation: since more and more complex workflow-based software systems are hierarchically constructed, we may have to strengthen the power of QWF-net to support hierarchical modeling and QoS analysis. Hierarchical QWF-net model should be capable of modeling a sub-net graph as a composite task.
– Developing non-Markovian QoS models (for instance semi-Markovian model) to support modeling of inconstant firing-rate/failure-rate of task: in this paper, it is assumed that every task has constant firing-rate/failure-rate. For further study, we are going to introduce inconstant firing-rate/failure rate to capture general distribution of firing-delay/TTF of task.
– Developing approximation methods: in this paper, the methods of calculating exact $E(\mathit{MS})/\sigma(\mathit{MS})/E(C)/\sigma$ $(\mathit{COST})/R$ are given. However, software engineers and practitioners may also need fast, concise and approximate methods for quick evaluation of system QoS. Therefore approximation methods are to be introduced in further study.
– Developing selection and scheduling algorithms: system managers sometimes have to select from a variety of available tasks to compose a workflow system and design its scheduling policy to fulfill user requirements. Based on analytical models proposed in this research, we are going to develop selection and scheduling algorithms guiding system managers to identify the optimal selection and scheduling solutions

References

[1] A. Ferscha, *Qualitative and Quantitative Analysis of Business Workflows using Generalized Stochastic Petri Nets*, Proceedings of Workflow Management – Challenges, Paradigms and Products, 1994. ACM press: Madison, Wisconsin, 1994, 222–234.

[2] W.R. Gilks, G.O. Roberts and S.K. Sahu, Adaptive Markov chain Monte Carlo through regeneration, *Journal of the American Statistical Association* (1993), 1045–1054.

[3] H. Niederreiter, *Random Number Generation and Guasi-Monte-Carlo Methods*, SIAM, Philadelphia, 1992.

[4] J. Li, Y. Fan and M. Zhou, Performance modeling and analysis of workflow, *IEEE transaction on Systems, man, and cybernetics-part A: Systems and humans* **34**(2) (2004), 229–242.

[5] J. Klingemann, J. Waesch and K. Aberer, *Deriving Service Models in Cross-Organizational Workflows*, ProceedingS of lst Workshop on Reasearch Issues in Data Engineering (RIDE), 1999, IEEE Computer Society Press: Boston, MA, 1999, 100–108.

[6] J. Cardoso, A. Sheth, J. Miller, J. Arnold and K. Kochut, Quality of service for workflows and web service processes, *Elsevier Transaction on web semantics* **1**(3) (2004), 281–308.

[7] J.-H. Lo, C.-Y. Huang, S.-Y. Kuo and M.R. Lyu, *Sensitivity Analysis of Software Reliability for Component-Based Software Applications*, Proceedings of 27th Annual International Computer Software and Applications Conference, 2003, IEEE Computer Society Press: Boston, MA, 2003, 500–505.

[8] K. Kanoun and T. Sabourin, *Software Dependability of a Telephone Switching System*, Proceedings of 17th International Symposium on Fault-tolerant Computing, 1987, 236–241.

[9] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, QoS-Aware Middleware for Web Services Composition, *IEEE Transaction on Software Engineering* **30**(5) (2004), 311–327.

[10] C. Lin, Y. Qu, F. Ren and D.C. Marinescu, *Performance Equivalent Analysis of Workflow Systems Based on Stochastic Petri Net Models*, Proceedings of 1st International Conference on Engineering and Deployment of Cooperative Information Systems (Lexcture notes in computer science, Vol. 2480), 2002. Springer:Berlin, 2002, 64–79.

[11] M.C. Jaeger, G. Rojec-Goldmann and G. Muhl, *QoS Aggregation in Web Service Compositions*, Proceedings IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005, IEEE Computer Society Press: Boston, MA, 2005, 181–185.

[12] M. Gillmann, J. Weissenfels, G. Weikum and A. Kraiss, *Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems*, Proceedings of 7th International Conference on Extending Database Technology (Lexcture notes in computer science, Vol. 1777), 2000, Springer:Berlin, 2000, 183–201.

[13] M. Gillmann, G. Weikum and W. Wonner, *Workflow Management with Service Quality Guarantees*, Proceedings of ACM SIGMOD International Conference on Management of Data, 2002, ACM press: Madison, Wisconsin, 2002, 228–239.

[14] T.B. Pinkerton, *Program Behavior and Control in Virtual Storage Computer Systems*, Technical Report 4, University of Michigan, 1968.

[15] C.V. Ramamoorthy, *The Analytic Design of a Dynamic Look Ahead and Program Segmenting System for Multiprogrammed Computers*, Proceedings of ACM National Conference, 1966, 229–239.

[16] R.Y. Rubinstein, *Simulation and the Monte Carlo Method*, Wiley, New York, NY, 1981.

[17] S.-Y. Hwang, H. Wang, J. Srivastava and R.A. Paul, *A Probabilistic QoS Model and Computation Framework for Web Services-Based Workflows*, Proceedings of 23rd International Conference on Conceptual Modeling (Lexcture notes in computer science, Vol. 3288), 2004, Springer:Berlin, 2004, 596–609.

[18] S. Gokhale et al., *Reliability Simulation of Component-Based Software Systems*, In *Proc. 9th Int. Symp. Software Reliability Engineering*, 1998, 192–201.

[19] S. Gokhale, W.E. Wong, K. Trivedi and J.R. Horgan, *An Analytical Approach to Architecture Based Software Reliability Prediction*, Proceedings of 3rd International Computer Performance and Dependability Symposium, 1998, IEEE Computer Society Press: Boston, MA, 1998, 13–22.

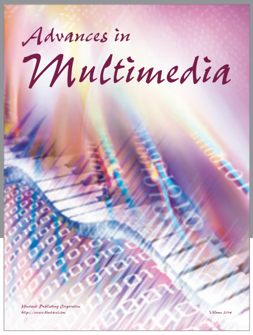
[20] Stroock and W. Daniel, *An Introduction to Markov Processes*, Springer, Newyork, 2005.

[21] S.S. Gokhale and K.S. Trivedi, *Reliablity Prediction and Sensitivity Analysis Based on Software Architecture*, Proceedings of 13th IEEE International Symposium on Software Reliability Engineering, 2002. IEEE Computer Society Press: Boston, MA, 2002, 64–78.

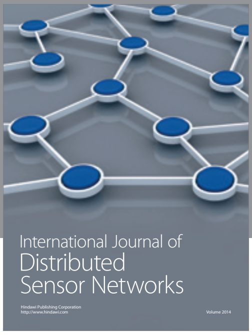
[22] W. van der Aalst and K. van Hee, *Workflow Management: Models, Methods, and Systems*, The MIT Press, 2002.

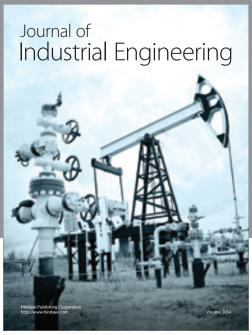
[23] W. Everett, *Software Component Reliability Analysis*, Proceedings of Symposium on Application-specific Systems and Software Engineering Technology, 1999, IEEE Computer Society Press: Boston, MA, 1999, 204–211.

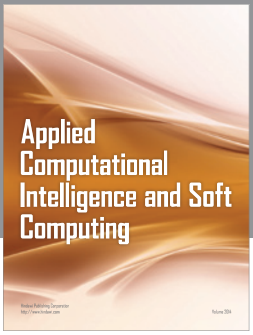
[24] Z. Tan, C. Lin, H. Yin, Y. Hong and G. Zhu, *Approximate Performance Analysis of Web Services Flow Using Stochastic Petri Net*, ProceedingS of the 3rd Grid and Cooperative Computing GCC Conference (Lexcture notes in computer science, Vol. 3251), 2004, Springer:Berlin, 2004, 193–200.

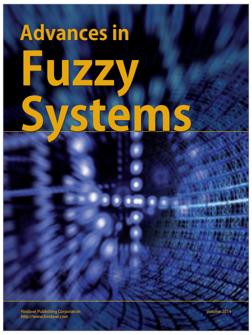

Advances in
Multimedia

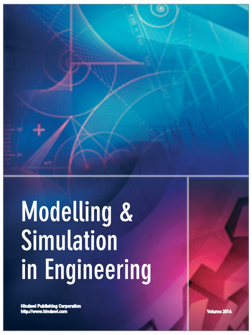
The Scientific
World Journal

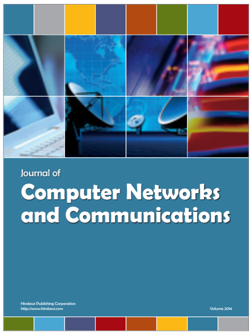

International Journal of
Distributed
Sensor Networks

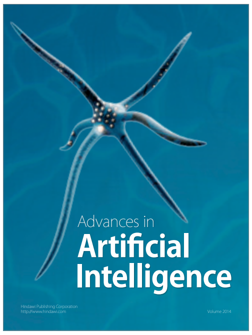

Journal of
Industrial Engineering

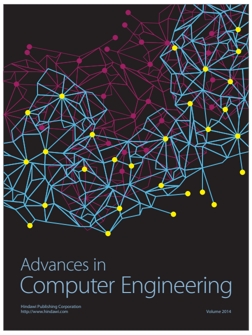

Applied
Computational
Intelligence and Soft
Computing


Advances in
Fuzzy
Systems

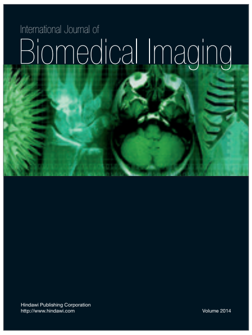

Modelling &
Simulation
in Engineering

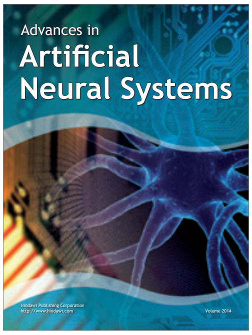

Journal of
Computer Networks
and Communications

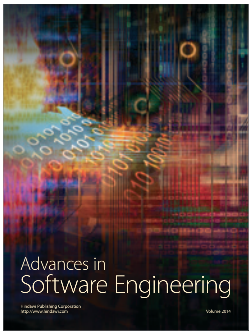

Advances in
Artificial
Intelligence


Advances in
Computer Engineering

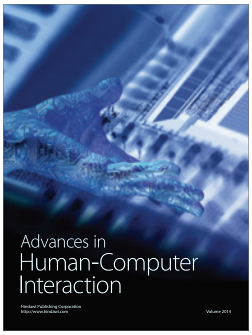
International Journal of
Computer Games
Technology

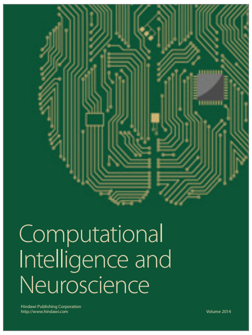

International Journal of
Biomedical Imaging


Advances in
Artificial
Neural Systems


Advances in
Software Engineering

Journal of
Robotics


Advances in
Human-Computer
Interaction


Computational
Intelligence and
Neuroscience

International Journal of
Reconfigurable
Computing

Journal of
Electrical and Computer
Engineering