To appear in IEEE Transactions on Services Computing, accepted 18 Dec 2019 (c) IEEE 2019

Runtime Verification of Business Cloud Workflow Temporal Conformance

Haoyu Luo, Xiao Liu, *Senior Member, IEEE*, Jin Liu, *Member, IEEE*,
Yun Yang, *Senior Member, IEEE*, and John Grundy, *Senior Member, IEEE*

Abstract—Business cloud workflows are often designed with multiple time constraints for timely response to business requests. To ensure on-time completion of workflow instances, workflow temporal conformance state needs to be constantly monitored and verified at runtime. Considering the fact that there are a large number of workflow instances running in a parallel fashion in many business scenarios, conventional verification approaches for time-related properties using such as temporal logic or timed Petri nets are not feasible due to the limitation of low efficiency at runtime. To address this issue, we propose a new approach to automated runtime verification of temporal conformance for parallel workflow instances in a cloud environment. In this paper, instead of using response time to verify temporal conformance of every single workflow as in conventional strategies, workflow throughput is employed as the performance measurement to efficiently monitor a large number of parallel workflow instances. On this basis we present a novel conformance verification strategy. This strategy considers the effect of time delay propagation in the cloud workflow systems to accurately verify workflow runtime temporal conformance. Our verification strategy is implemented in a prototype cloud workflow system and the evaluation results show that it outperforms the state-of-the-art workflow temporal verification strategy.

Index Terms—runtime verification; business workflows; temporal conformance; on-time completion; cloud computing;

1 Introduction

Business workflows are pervasive in many large-scale business applications such as bank transactions and securities exchange. A notable characteristic of business workflow applications is that there are usually a large number of workflow instances running in a parallel and distributed fashion, where each instance represents a business request. Given the requirement in the processing of concurrent business requests, cloud computing that can provide cost-effective scalable resources is an ideal host environment for running large numbers of workflow instances.

Many large enterprise workflows have to meet multiple time constraints in order to achieve timely completion of business goals. However, always on-time completion is not possible in real business scenarios full of uncertainties. It is also not always necessary because most customers can have some level of endurance for small time delays. Therefore, instead of "hard deadlines", "soft deadlines" are commonly used for business workflows planning, executing and monitoring. For example, instead of specifying "every workflow must be completed within 5 minutes" as a hard deadline, we can specify "90% of workflows must be completed within 5 minutes" as a kind of soft deadline in QoS (Quality of Service) specification for business workflow application. "90%" as an example target on-time completion rate can be decided by the enterprises based on their target customer satisfaction rate or economical concerns.

However, due to the dynamic nature and uncertainties that exist during the running of workflows in the cloud, temporal violations often occur. Here, "temporal violation" means an intermediate violation of time constraint (i.e., time delay) that can be fixed locally to ensure overall timely completion. A large number of unpredictable temporal violations will seriously jeopardize the timely completion of massive concurrent user requests, which may lead to not only the deterioration of user satisfaction but also the expiration of results, even the deal's collapse.

To deliver satisfactory on-time completion rate of time-constrained workflows, workflow temporal conformance needs to be constantly monitored and verified. Here workflow temporal conformance denotes the consistency between runtime workflow execution states and build-time temporal QoS specification. The approach for monitoring workflow temporal conformance is called *workflow temporal verification*, which is conducted to guarantee satisfactory temporal QoS in workflow systems. Given a workflow lifecycle, a temporal verification framework consists of three components, viz, temporal constraint setting [31], temporal consistency verification [32] and temporal violation handling [33]. Temporal constraint setting assigns time constraint for workflow activities according to the deadline specified in QoS specification. Temporal consistency verification monitors and verifies whether workflow temporal behavior complies as expected, and determines whether a temporal violation occurs or not. If a temporal violation is detected, the current temporal behavior needs to be

- *H. Luo is with the School of Computer Science, South China Normal University, Guangzhou, 510631, P.R. China. E-mail: hluo@m.scnu.edu.cn.*
- *J. Liu is with the School of Computer Science, Wuhan University, Wuhan, 430072, P.R. China. E-mail: jinliu@whu.edu.cn.*
- *X. Liu is with the School of Information Technology, Deakin University, Geelong, VIC 3216, Australia. E-mail: xiao.liu@deakin.edu.au.*
- *Y. Yang is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia. E-mail: yyang@swin.edu.au.*
- *J.C. Grundy is with the Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia. E-mail: John.Grundy@monash.edu.*

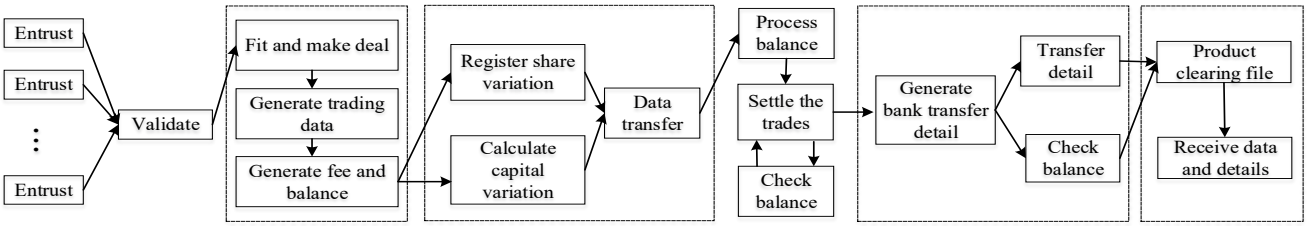


Fig. 1. A securities exchange business process

adjusted by temporal violation handling strategies.

Workflow temporal verification can be categorized as build-time verification and runtime verification. Formal methods based on Petri nets or event algebra [1], [2] mainly focus on process modeling and verification at workflow build-time. These static approaches cannot detect and automatically respond to failures when runtime temporal conformance states violate the time constraint specifications. Runtime verification aims to check whether a run of the system satisfies a given correctness property at any point in time [3]. Existing research on workflow runtime verification focuses on monitoring the running state of a workflow instance and determining whether it complies with time constraint specifications. However, these approaches that are designed to monitor single runs of the system have to be repeated thousands of times to monitor thousands of parallel workflow instances. It is prohibitively expensive to implement parallel workflow monitors as the overhead increases dramatically.

Limitations of current business workflow verification approaches lead us to the following research questions:

RQ1: *What do we need to measure for the purpose of monitoring the temporal conformance state of a large number of parallel workflow instances at runtime?*

RQ2: *Where or when can we most efficiently monitor the workflow temporal conformance state?*

RQ3: *How do we effectively verify workflow temporal conformance state?*

In this paper, we propose a novel runtime verification approach to monitor and verify temporal conformance of parallel business workflows running in the cloud. We aim to provide an efficient and effective method to monitor workflow behavior at runtime, which helps to achieve the target workflow on-time completion rate. Specifically:

1) For the first research question, to reduce the verification overhead, instead of using the response time of workflow activities to monitor every single workflow instance, *workflow throughput* is employed as the performance measurement for describing the behavior of a large batch of parallel workflow instances.

2) For the second research question, to achieve high efficiency, we adapt an efficient throughput based temporal checkpoint selection method from our earlier work in [4] to select only a small but sufficient number of time points along the workflow system execution timeline as temporal checkpoints where temporal verification needs to be conducted to check the temporal conformance state.

3) For the third research question, a novel temporal verification strategy is proposed for the temporal conformance monitoring and verification purpose. This strategy considers the propagation of time delays in the cloud workflow environment that enables us to produce a much more accurate verdict of temporal conformance for a large number of parallel cloud workflow instances.

Note that to keep consistency with other related work using the above mentioned temporal verification framework, the proposed approach deals with temporal verification of business workflow in a cloud environment in this paper. In terms of the method and models proposed by our approach, they are in fact applicable in a generic computing environment.

Our strategy is implemented in a prototype cloud workflow system. Experimental results show that our strategy outperforms the state-of-the-art workflow temporal verification strategy for large-scale parallel workflow instances.

2 Problem Statement and Analysis

2.1 A Common Scenario in Securities Exchange

Securities exchange is a typical instances-intensive business workflow. In general, a securities exchange workflow can be divided into online trading and offline settlement.

Online trading processes are completed in several minutes after a client performs dealing operations on the terminal. Client entrustment will be sent to the stock market to clinch a deal. Then the fitting results are recorded into the database in Securities Corporation and feed back to the client. So far, the deal is technically completed but the share is not legally owned by the client until offline settlement is finished after the market is closed to all clients.

An offline settlement process is responsible for archiving and clearing all the completed online deals. During offline settlement, a large number of workflow instances are initialized over a period of time, each instance represents an online deal record. Figure 1 illustrates the simplified securities exchange workflow of Chinese Shanghai A-Share Stock Market, which involves a few steps as workflow activities, including "register shares variation", "calculate capital variation", "settle the trades", "transfer capital" and so on. Some steps of the workflow instance are executed concurrently. The process is eventually completed only if Clearing Corporations finish stock clearing and the balance of all the capital transferred is zero.

In the scenario of securities exchange, on-time completion is a key non-functional requirement that should be satisfied. Similar requirements for temporal QoS are common in many other soft real-time systems. For example, an e-commerce website needs to process hundreds of thousands of orders every day. Massive parallel workflow instances

for order processing are initialized within a short time and need to be completed in a reasonable duration for user satisfaction. But unlike some hard real-time systems (e.g., aerospace control system) that must adhere to strict execution deadlines, violating a time constraint can be tolerated in soft real-time system providing that this happens with a sufficiently low probability. The threshold for the percentage of deadline misses (namely target on-time completion rate) depends on specific scenarios.

2.2 Problem Analysis

In the previous scenario, the number of entrustments of securities transactions can reach several millions per second at peak time. Therefore, automated run-time verification is essential for the real-time monitoring purpose. Besides, the durations of business workflow activities are often very short so that temporal non-conformance state should be detected and then be handled as soon as possible. This requires that the temporal verification model is sensitive to time delays and can accurately determine the transient temporal state of workflow instances.

When dealing with the monitoring of massive parallel workflow instances, scalability becomes a critical issue. Response-time based strategies which are designed for monitoring single large-scale scientific workflow instance use the response time of individual workflow activities as the monitoring target, and one monitoring service is dedicated to one workflow instance [6]. Therefore, the temporal verification overhead will grow linearly to the number of parallel workflow instances. This is unacceptable in instance-intensive business workflows where hundreds of thousands of workflow instances can be running in parallel. In this case, instead of the response time, a new measurement for a batch of workflow instances is needed to ensure high scalability of temporal verification.

Therefore, an efficient and effective runtime temporal verification is required in monitoring temporal conformance state of a large number of parallel workflow instances running in large-scale business cloud system.

3 Definitions

In this section, we analyze and address the first two research questions.

RQ1: *What do we need to measure for the purpose of monitoring the temporal conformance state of a large number of parallel workflow instances at runtime?*

In runtime verification, checking whether a run of the system meets a correctness property is performed using a decision procedure, called monitor. It has been suggested by Nutt [7] that the most important questions to be answered before attempting to monitor a system are 'what to measure' and 'why the measurement should be taken'.

Response time and throughput are two primary measurements of workflow applications from the performance perspective [8]. While response time is to measure the duration of a workflow activity or the makespan of a workflow instance, throughput measures the number of workflow activities that have been finished per time unit. As mentioned above, response time is not applicable for monitoring a large number of parallel workflow instances due to the large monitoring overhead. It is also unnecessary to check the intermediate temporal state of every single workflow instance since our ultimate goal is to achieve the target on-time completion rate for the whole instances. Therefore, workflow throughput is employed as the performance measurement for business cloud workflows.

In addition, measuring workflow throughput by counting the number of completed workflow activities is not reasonable because the completion of activities with different durations have different contributions to the on-time completion of the whole workflow instances. The authors in [4] have proposed a new definition of workflow throughput to reflect the contributions of completing individual activities to the on-time completion of all workflow instances. Next, before we present the workflow throughput definition, some basic workflow time attributes are introduced.

Business workflow is made up of a set of activities in partial order. We denote the ith activity of a business workflow as a_i. The maximum, mean, minimum, expected and runtime completion duration of a_i is denoted as $D(a_i)$, $M(a_i)$, $d(a_i)$, $E(a_i)$ and $R(a_i)$ respectively. Accordingly, WF_i is a workflow instance with its maximum, mean, minimum, expected and runtime completion duration denoted as $D(WF_i)$, $M(WF_i)$, $d(WF_i)$, $E(WF_i)$ and $R(WF_i)$ respectively.

Definition 1 (Workflow Throughput). *Given a batch of q parallel instances $\{WF_1, WF_2, \ldots, WF_q\}$ of business workflow WF which starts at system time S_0, the completion of workflow activity a_{ij} (namely the jth activity of WF_i) contributes to the completion of the entire collection of workflows with a value of $M(a_{ij})/T$ where $T=\sum_{i=1}^{q} M(WF_i)$. Here, we assume that at the current observation time point S_t, the set of new completed activities from the last nearest observation time point S_{t-1} is denoted as $a\{\}|_{S_{t-1}}^{S_t}$, then the system throughput is defined as $TH|_{S_{t-1}}^{S_t} = M(a\{\}|_{S_{t-1}}^{S_t})/T$.*

Definition 1 presents how much of those activities completed during the last observed time unit contributes to the completion of all workflow instances. Correspondingly, workflow throughput constraints need to be defined to match the use of workflow throughput, which is the expected accumulated workflow throughput that should be achieved by a specific system time point.

Definition 2 (Workflow Throughput Constraints). *Given the same batch of workflow instances as defined in Definition 1, the throughput constraint assigned at system point S_t is denoted as $THCons|_{S_0}^{S_t}$ which means that the accumulated throughput $\sum_{i=1}^{t} TH|_{S_{i-1}}^{S_i}$ should be no less than the value of the assigned throughput constraint. The value of $THCons|_{S_0}^{S_t}$ is decided by the deadline assignment strategy.*

Work in [5] presents a representative deadline assignment strategy. This strategy employs a queueing model to predict workflow activity durations, and then calculates the expected percentage of workflow instances completion as local throughput constraint at any system time point.

RQ2: *Where or when can we most efficiently monitor the workflow temporal conformance state?*

Workflow temporal verification is the major approach for delivering satisfactory temporal QoS in business cloud systems by monitoring workflow temporal conformance state at runtime. Given the throughput constraint defined above, workflow temporal verification is to check whether the target throughput constraints can be satisfied or not at a specific system time point, also known as a *throughput checkpoint* or *checkpoint* for short.

Theoretically, any time point along the system timeline can be a checkpoint. However, since in practice monitoring is usually conducted discretely along the system timeline, there is normally a basic time unit (*e.g.,* one minute) denoted as bt, which defines the frequency for updating workflow time attributes for the monitoring purpose. Throughput checkpoints are selected based on these monitoring points that are candidate throughput checkpoints.

Definition 3 (Candidate Throughput Checkpoints). *Given the same batch of workflow instances as in Definition 1, a system time point S_t along the workflow execution timeline is a candidate throughput checkpoint if $S_t - S_{t-1} = k * bt$ (k = 1,2,3 ...).*

By definition 3, candidate throughput checkpoints are statically set before workflow execution, a checkpoint selection strategy is used to select the real checkpoints where temporal verification needs to be conducted at runtime. In this paper, since our focus is on temporal verification, we just adopt a representative throughput based checkpoint selection strategy as proposed in [9].

Throughput based Checkpoint Selection Strategy: *Given the same collection of workflow instances as in Definition 1, the rule of throughput based checkpoint selection strategy is defined as follows: given the candidate throughput checkpoint S_p, if $TH|_{S_{p-1}}^{S_p} < THCons|_{S_{p-1}}^{S_p}$, S_p is selected as a checkpoint. Otherwise, S_p is not selected as a checkpoint. Here, $TH|_{S_{p-1}}^{S_p}$ is the runtime throughput during the time points S_{p-1} and S_p , $THCons|_{S_{p-1}}^{S_p}$ is the expected percentage of completion between S_{p-1} and S_p.*

In summary, in our approach workflow time attributes are monitored at each candidate checkpoint and temporal verification is conducted at each selected checkpoint to monitor and verify temporal conformance state.

It should be noted that some definitions in this paper may be slightly different from others presented in some of our earlier publications (e.g., in [5] and [9]) when defining the same concept in the scenario of business workflow temporal verification. The main reason for this is the different and focus of concern varies in the different research works. For example, the definition of "workflow throughput" appears with a W array in [5] and [9], because the influence of the workflow structure is considered in the proposed strategies. While in order to focus on the workflow monitoring and verification strategy itself, complex workflow structures are pre-processed into sequential structure in this paper, thus W array is removed in the related definitions. We are continuously improving and evolving the models and definitions, which may cause variations among these different research works.

4 Time Delay Propagation Analysis

In the next two sections, we describe our new solution to solve the third research question, namely:

RQ3: *How do we effectively verify workflow temporal conformance state?*

Based on the answers to the first two research questions, we can see temporal conformance state of parallel workflow instances needs to be verified using a *throughput based temporal verification strategy*. In this paper, we propose a throughput conformance verification strategy which considers the effect of time delay propagation in cloud workflow systems to produce more accurate verdicts of workflow temporal conformance. In this section, we analyze the effect of time delay propagation.

Time delay propagation is common during the execution of parallel workflow instances in business cloud workflow systems due to resource sharing and the temporal dependencies among workflow activities. As a large number of workflow instances are processed in parallel, they have to be queued up waiting for execution on a limited number of cloud services. Once an activity violates its local temporal constraint (*i.e.,* the actual response time exceeds the expected time), all other activities in the same queue will have to wait longer to be executed. As a consequence, some activities are likely to violate their own temporal constraints as well. Meanwhile, as workflow activities need to be executed in a partial order, delays to one activity will postpone the start time of subsequent activities of the same workflow instance waiting on other cloud services. As a result, activities of other workflow instances waiting on those cloud services may also be delayed.

Therefore, a few time delays may result in massive temporal violations since time delays can propagate among both subsequent activities of the same workflow instance and activities of other parallel workflow instances that are competing for the same resources. Finally in the worst case, many parallel workflow instances may violate their final deadlines, which results in the failure of achieving the target on-time completion rate.

Such a of time delay propagation process is similar to the famous "*Butterfly Effect*". Neglecting time delay propagation may lead to false negative verification result since some potential violations caused by the propagation effect are not considered and thus cannot be handled in time. Without timely handling of temporal violations, the accumulated time delays will be impossible to be compensated and eventually lead to the failure of on-time completion.

For example: Assuming that the target on-time completion rate of workflow instances is set as 90%. At checkpoint S_p, the instant temporal conformance state β% is verified to be 95%. Obviously, the result indicates that the target on-time completion rate can be achieved according to the workflow execution progress until S_p.

However in fact, due to the "*Butterfly Effect*", the detected temporal violations at S_p may introduce more potential temporal violations which will appear after S_p. Although these potential violations can eventually be detected at the successive checkpoints, the system may not be able to fully compensate the accumulated time delays, especially when the checkpoints are close to the final deadline. When the implicit effect of time delay propagation is considered, the actual temporal conformance state $\beta'\%$ at S_p may be only 85%, rather than 95%. In this case, temporal violation handling strategies need to be triggered to timely compensate for the time delays.

To investigate how time delays propagate in the cloud system at workflow runtime, we first model the queueing and execution process of parallel workflow instances in the cloud system.

4.1 Queueing Model for Cloud Services

At workflow runtime, a large number of parallel instances of a business process are initialized in a short time (the middle tier in Figure 2). Each instance has to be executed step by step according to the business logic. Since the number of parallel workflow instances is normally much larger than the dedicated cloud services, workflow activities have to queue up on limited services. In Figure 2, workflow activities with the same colour represent the same kind of business activities and queue in the same queueing system with dedicated cloud services. Noted that for ease of discussion, we assume that each queueing system is dedicated to only one type of cloud service, and one type of cloud service is only for one type of activity. So if a business process consists of k atomic business activities, there will be k queueing systems. These queueing systems with various queuing features together form a well-connected queueing network for the cloud workflow system.

When a workflow instance arrives at the first queueing system, it will be either served immediately or waiting for service, and then leaves the queueing system for the next one after service. We employ $M/G/m/m+r$ model to formulate the behavior of the first queueing system.

Definition 4 (M/G/m/m+r Queueing Model for Cloud Services). *$M/G/m/m+r$ model hypothesizes that the inter-arrival time of service requests arrives according to a Poisson process with rate λ, while execution time of activities are independent and identically distributed random variables that follow a general distribution model with a mean value of μ. The queuing system contains m cloud services and service order is First Come First Service. The maximum number of activities in the queueing system including those being serviced is $m + r$, where r is the buffer size for incoming requests.*

With this queueing model, we can obtain performance related attributes that accurately reflect the queuing process, such as mean waiting time, mean response time and mean number of activities in the queuing system [10].

Furthermore, since the service time of the first activity follows a general distribution in the first queueing system, the inter-arrival time of activities in the next $k-1$ queuing systems will follow a general distribution model as well.

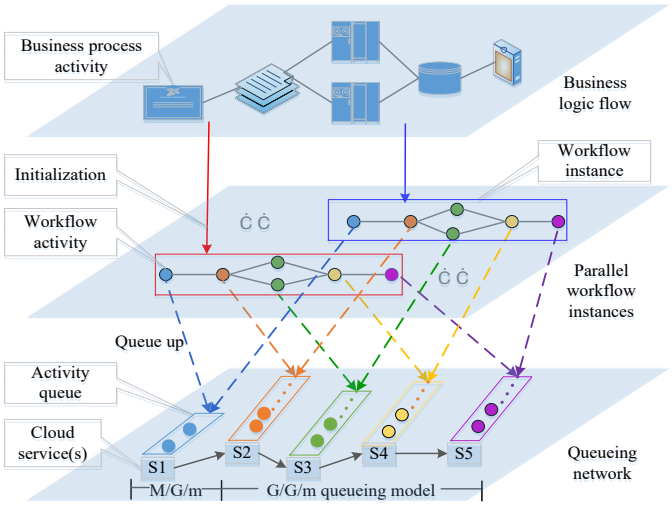


Fig. 2. Queueing models for cloud services

Hence the queuing model for these $k-1$ queueing systems is $G/G/m/m+r$.

Definition 5 (G/G/m/m+r Queuing Model for Cloud Services). *$G/G/m/m+r$ model hypothesizes that the inter-arrival time of service requests and execution time of activities are independent and identically distributed random variables that follow a general distribution with a mean value of μ. The queuing system contains m cloud services and the service order is First Come First Service. The maximum number of activities in the queueing system including those being serviced is $m + r$, where r is the buffer size for incoming requests.*

However, given the variations and complexity of the $G/G/m/m+r$ model, it is difficult to accurately measure the performance of the queueing system. In this paper, we adopt an approximate solution presented by Atmaca *et al.* in [11] for $G/G/m/m+r$ queuing system to reduce the complexity of the problem.

4.2 Time Delay Propagation in a Workflow Queueing Network

The response time of a workflow activity in a queueing system can be divided into queueing time and execution time. Time delay occurred in the queueing period is called passive delay and the time delay occurred in the execution period is called active delay.

Active time delays occur for various reasons. Firstly, the dynamic nature of cloud services makes performance fluctuation inevitable, which may lead to response time delay of workflow activities. One active time delay may cause a single workflow instance to violate its final deadline. However, the propagation of time delays may cause temporal non-conformance and jeopardize the timely completion of a large number of parallel workflow instances in the same queueing network.

Time delay propagation can be classified into three different stages according to its level of effect on temporal conformance state. Here, we define some basic annotations: QS_j is a queueing system with n cloud servers (*i.e.*, cloud server instances) and an average of l activities in the queue. a_{ij} is the activity of WF_i in QS_j with its expected duration

donated as $E(a_{ij})$. TD_{ij} is the active time delay of a_{ij} during the execution in QS_j. The passive time delay of workflow activity is denoted as Δt. In the following, we analyze the propagation process in queueing network when time delay TD_{ij} is occurred.

(i) Propagation effect on a single workflow instance

For workflow instance WF_i, the subsequent activities of a_{ij} will postpone their start time by TD_{ij} since they have to wait in the queue until a_{ij} is completed. If all the other activities are timely completed, the overall response time of this workflow instance will be delayed by TD_{ij}.

(ii) Propagation effect on workflow activities in the same queueing system

For a queueing system with n servers, a newly arrived activity can be executed immediately if the queue is empty, otherwise it has to wait for service. Activities in the queueing system can be divided into three categories according to their arrival time. (1) The activities that arrive at the queue before a_{ij} starts execution; (2) The activities that arrive at the queue during the execution of a_{ij}; and (3) the activities that arrive at the queue after $a_{ij}$is completed.

If $n = 1$, for the first kind of activities, they have to postpone their start time by TD_{ij}. Thus, their potential response time delay $\Delta t = TD_{ij}$. For the second kind of activities, Δt increases with arrival time. The maximum value of Δt is denoted as ΔT_{max}. $\Delta T_{max} = \Delta l * E(a_{ij})$, where Δl is the difference of the queue length before and after a_{ij} is completed. For the third kind of activities, Δl is equal or less than ΔT_{max}.

If $n > 1$, the number of available servers during TD_{ij} is $n - 1$ and increases back to n after a_{ij} is completed. For the first kind of activities, $\Delta t = E(w(n-1)) - E(w(n))$, where $E(w(n))$ denotes the expected waiting time of activities in the queue when the number of servers is n. For the second kind of activities, since $E(w(n-1)) - E(w(n))$ decreases with the increasing number of servers, $E\big(w(n-1)\big) - E\big(w(n)\big) \leq TD_{ij}$. Hence $\Delta t \leq TD_{ij}$. For the third kind of activities, response time delays decrease in the order that they arrive at the queue, since the length of the queue gradually decreases to normal once a_{ij} is completed. The maximum value of Δt is denoted as ΔT_{max}. $\Delta T_{max} = \Delta l * E(a_{ij})$, where Δl is the difference between the expected queue length and actual queue length.

(iii) Propagation effect on subsequent queueing systems

The underlying cloud services of a cloud workflow system formed a network of queueing systems as shown in Figure 2. The above two stages describe the time delay propagation among temporal dependent workflow activities, while the propagation among queueing systems in the queueing network is much more complicated as much more workflow activities and workflow instances are involved. Therefore, workflow throughput instead of response time is employed to measure the propagation effect on subsequent queueing systems.

According to *Definitions 1* and *2*, the throughput of queueing system QS_j is defined as $TH_j|_{S_{t-1}}^{S_t} = w_j * TH|_{S_{t-1}}^{S_t}$, where w_j is the activity duration weight of a_{ij} in the overall duration of WF_i. The corresponding throughput constraint is defined as $THCons_j|_{S_{t-1}}^{S_t} = w_j * THCons|_{S_{t-1}}^{S_t}$.

When the execution time of one or more activities in QS_j exceeds the expected time, a throughput constraint may be violated as the number of completed activities within a basic time unit is less than expected. We denote the gap between the actual throughput and the throughput constraint as ΔTH_j. $\Delta TH_j = THCons_j|_{S_{t-1}}^{S_t} - TH_j|_{S_{t-1}}^{S_t}$. The detected throughput violation ΔTH_j means that the number of activities leaving QS_j in unit time is less than expectation. Since the arrival rate of QS_{j+1} equals to the leaving rate of QS_j, the throughput of QS_{j+1} will be less than the expected value in the following short period for those late arrived activities. Specifically:

If $l_{s_{t-1}} = 0$, namely the queue of QS_{j+1} is empty at observation time point S_{t-1}, the difference between the number of completed activities and expected value in QS_{j+1} is equal to the one in QS_j. Based on Definition 3, $\Delta TH_{j+1} = \Delta TH_j * (E(a_{j+1})/(E(a_j)))$.

If $l_{s_{t-1}} > 0$, the queue length of QS_j decreases since the arrival rate is less than expected. When the queue length decreases to 0 before arrival rate returns back to normal, $\Delta TH_{j+1} = 0$, which means throughput violation at QS_j has no effect on the throughput of QS_{j+1}. Otherwise, $\Delta TH_{j+1} < \Delta TH_j * (E(a_{j+1})/(E(a_j)))$.

Therefore, given the detected throughput constraint violation ΔTH_j in QS_j, the maximum propagation effect on the throughput of QS_{j+1} can be represented as:

$$ThrEffMax_j = \Delta TH_j * \frac{E(a_{j+1})}{E(a_j)} \quad (1)$$

The maximum propagation effect on the throughput of the overall workflow system is the accumulation of the maximum effect on its sequential queueing systems, given by:

$$PropEffMax_j = \sum_{i=j}^{n} ThrEffMax_i \quad (2)$$

Clearly, there is a gap between the theoretical maximum and the actual propagation effect that happens at runtime. Moreover, since the propagation effect fluctuates constantly in the dynamic cloud environment, it is difficult to measure the actual effect at runtime. For example, the effect of time delay may be decreased for the reason that some response time delays can be automatically compensated by time redundancy produced in the execution period [6]. Therefore, we need to constantly update the estimated propagation effect based on the monitoring results so that the propagation effect in the model gets closer to actual situation.

5 A Novel Runtime Verification Strategy

In this section, we first introduce a new propagation-aware throughput conformance model, and then present a

throughput conformance verification strategy.

5.1 Propagation-aware Throughput Conformance Model

To define a runtime throughput conformance model, we need to measure how much throughput has been completed by the current checkpoint and estimate how much throughput can be completed between the current checkpoint and final deadline. The former can be easily obtained based on the definition of runtime workflow throughput. The estimated throughput is decided by the remaining time and the durations of subsequent activities.

Definition 6 (Estimated Workflow Throughput). *Given the same collection of q parallel workflows as in Definition 1, its fixed-time deadline denoted as $F(WF)$ and its upper-bound constraint $U(WF)$, at throughput checkpoint S_p, the expected workflow throughput for the remaining time is defined as:*

$$Exp\left(TH|_{S_p}^{F(WF)}\right) = TH|_{S_p}^{F(WF)} * \frac{q * U(WF) - R\left(a\{\}|_{S_0}^{S_p}\right)}{E\left(a\{\}|_{S_p}^{F(WF)}\right)} \quad (3)$$

Besides the above two explicit factors, the implicit propagation effect of time delays has to be included in the throughput conformance model. Although some minor temporal violations will not result in a throughput non-conformance state at current checkpoint, the gradually accumulated propagation effect of time delays may eventually cause the failure of timely completion of large collection of parallel cloud workflow instances.

Based on the above discussion, our novel runtime throughput conformance model is proposed as follows:

Definition 7 (Propagation-aware Throughput Conformance Model). *Given the same collection of q parallel workflow instances in Definition 1 and its fixed-time deadline denoted as $F(WF)$, at a throughput checkpoint S_p, it is said to be of α% conformance if:*

$$F(\lambda_\alpha) = TH|_{S_0}^{S_p} + Exp\left(TH|_{S_p}^{F(WF)}\right) - \sum_{i=1}^{n} PropEffMax_i - \sigma_p \quad (4)$$

Where λ_α is defined as α% confidence percentile with the cumulative standard normal distribution function of $F(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{\mu_i+\lambda\sigma_i} e^{-(x-\mu_i)^2/2\sigma_i^2}\,dx = \alpha\%$. $TH|_{S_0}^{S_p}$ is the current runtime throughput until S_p, $Exp\left(TH|_{S_p}^{F(WF)}\right)$ is the expected workflow throughput for the remaining time. $\sum_{i=1}^{n} PropEffMax_i$ is the total propagation effect caused by the throughput constraint violated queueing systems which are detected at S_p.

σ_p in *equation (4)* is the accumulated propagation effect caused by the throughput constraint violated queueing systems which are detected at the checkpoints before S_p.

$$\sigma_p = \sum_{k=1}^{p}\sum_{j=1}^{n} \Delta TH'_{kj} * \frac{E(a_{j+1})}{E(a_j)} \quad (5)$$

TABLE 1
THROUGHPUT CONFORMANCE VERIFICATION STRATEGY

Input:	Target on-time Completion rate θ% The workflow runtime throughput state at a candidate checkpoint S_p
Output:	Throughput conformance state at S_p
Step1:	**Throughput-based checkpoint selection** **If** $TH|_{S_{p-1}}^{S_p} < THCons|_{S_{p-1}}^{S_p}$ S_p is selected as a checkpoint, continue to **Step 2**; **Else** **Break** until the next candidate checkpoint S_{p+1}.
Step 2:	**Throughput Conformance Verification** Calculate workflow throughput: (1) the completed workflow throughput until S_p. (2) the expected remainding throughput from S_p to $F(WF)$. Calculate throughput propagation effect: (1) the newly generated propagation effect between S_{p-1} and S_p (2) the accumulated propagation effect until S_{p-1}. Given the throughput conformance model, certificate the throughput conformance state α% at checkpoint S_p **If** $\alpha\% \geq \theta\%$ **Break**. **Else** report a detected potential temporal violation.

Where $\Delta TH'_{kj}$ is the actual propagation effect at S_p which is initially caused by QS_j and detected at S_k.

In general, α% conformance is a probability confidence for on-time completion. It is used to measure the current service quality comparing with a target on-time completion rate θ%. θ% is an agreed negotiation result between the user and the service provider on the service quality.

5.2 Throughput Conformance Verification Strategy

We depict the throughput conformance verification strategy in Table 1. This strategy consists of two steps, namely checkpoint selection and throughput conformance verification at the selected checkpoints.

The checkpoint selection strategy determines whether a candidate checkpoint (*e.g.*, S_p) should be selected as a throughput checkpoint in accordance with the current workflow runtime throughput state and throughput constraint. If S_p is selected as a checkpoint, throughput conformance verification (step 2) is required. Otherwise, the strategy will move on until the system time arrives at the next candidate checkpoint. Step 2 verifies throughput conformance state at checkpoints using the proposed propagation-aware throughput conformance model. The aim is to check whether the current throughput conformance state α%, as defined in Definition 7, is no less than the target on-time completion rate θ% (namely target throughput conformance). If the throughput conformance state holds true (*i.e.*, $\alpha\% \geq \theta\%$), nothing needs to be done. Otherwise, a detected potential temporal violation is reported to the workflow system and violation handling strategy will be triggered.

6 IMPLEMENTATION

SwinFlow-Cloud[1] is a prototype cloud workflow system

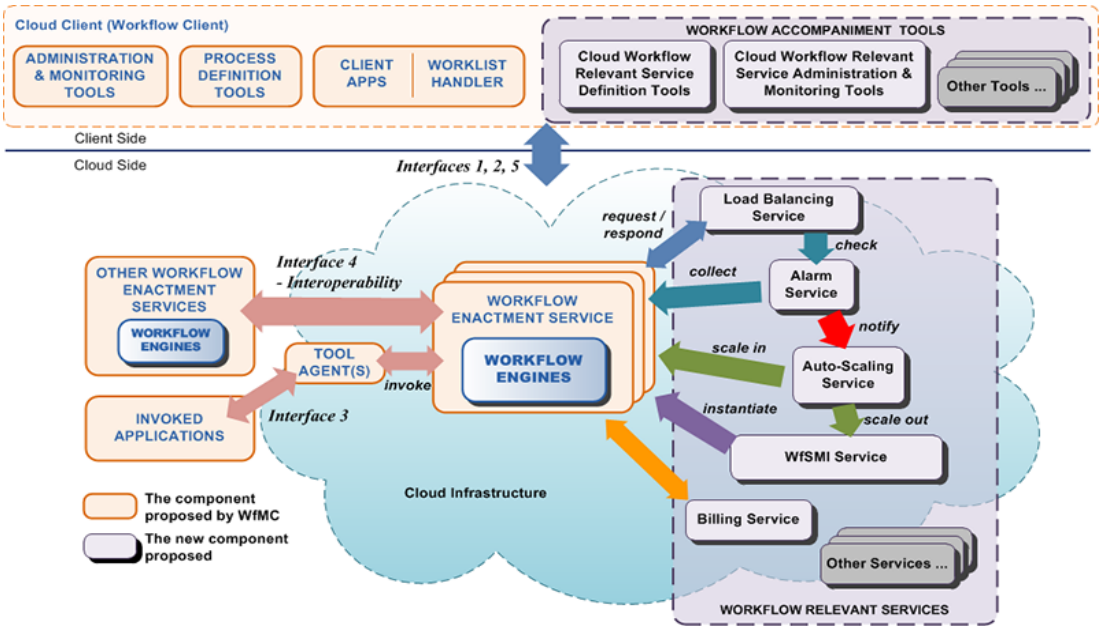


Fig. 3. SwinFlow-Cloud System Architecture [44]

designed to support the running of a large number of parallel business processes. As illustrated in Figure 3, adopting the client-cloud architecture (an extension of WfMC[2] proposed workflow reference model), SwinFlow-Cloud enables workflow users and administrators to easily create, run, and monitor workflow instances, and to make use of the powerful cloud computing infrastructures for running a large number of parallel workflow instances. The client side mainly consists of conventional workflow management tools and new workflow accompaniment tools. The cloud side is a group of scalable virtualized workflow engines and service components, it retains all the functional runtime service components of the traditional workflow reference model, and introduces many new non-functional service components which are defined as the workflow relevant services.

Here, we illustrate the process of cloud workflow temporal verification in SwinFlow-Cloud. The whole process, namely the temporal verification framework [12], includes three basic steps: temporal constraint setting, temporal conformance monitoring, and temporal violations handling. At the build-time stage, users can use the temporal constraint setting component, part of the *Cloud Workflow Relevant Service Definition Tools* on the client side, to determine the workflow deadlines and target on-time completion rates which are parts of the service contracts.

At the runtime stage, the start time, end time and other runtime attributes of workflow activities are logged by the system. The temporal conformance monitoring component, part of the *Cloud Workflow Relevant Service Administration and Monitoring Tools*, consists of a temporal checkpoint selection component and temporal verification component. The temporal checkpoint selection component can constantly or periodically read in the time attributes of workflow activities from the real-time system logs and use a checkpoint selection strategy to determine where temporal verification is needed. If a temporal checkpoint is selected, the temporal verification components will use the temporal conformance model to determine whether the workflow instances under monitoring are currently in a temporal non-conformance state or conformance state, *i.e.*, whether an intermediate temporal violation has occurred or not. If a temporal violation has been detected, the temporal violation handling component, a part of *Workflow Relevant Services* at the cloud side, will be triggered. Based on the level of temporal violations, corresponding handling strategies such as workflow rescheduling and resource recruitment [13] will be implemented by the violation handling component. However, if the temporal violation cannot be handled automatically due to critical errors, especially those out of the control of the workflow system *e.g.*, user input data is corrupted or the network connection at the client side is broken, an error message will be sent out to both the system administrators and users for manual intervention.

It should be noted that our work presented in this paper can work with or without the existence of cloud workflow systems. As long as the business processes are running in the cloud and data on activity runtime durations can be collected, workflow temporal verification can be conducted. However, with cloud workflow systems, workflow temporal verification can be easily integrated into the existing monitoring service. Meanwhile, the collection of activity runtime durations does not need to be conducted specially for temporal verification purpose as the workflow engine already have all of them for workflow scheduling purpose. Therefore, it is more efficient and cost-effective with a cloud workflow system.

1. http://www.xuanqiyun.com/swinflowcloud/
2: http://www.wfmc.org/standards/model.htm

TABLE 2
EXPERIMENTAL SETTINGS

Workflow instance size	Workflow instances with 15, 20, 25 activities are respectively tested.
Number of workflow instances	The number of parallel workflow instances increases from 3000, 6000 to 10000.
Workflow structure	Every workflow instance has the same workflow structure with sequential activities
Activity durations	The execution time of activity is generated using the simulation environment *Simulink* which is integrated with MATLAB. The mean execution time of activities are randomly selected from a range of 5 to 15 seconds and the coefficient of variation is 0.2.
Noise setting	Noise level: 0%, 25% and 50% of the execution time. Noise range: 5% of the activities.
Temporal constraints	The target on-time completion rate is set as 90%. The upper bound of temporal constraints is set as 120% of the mean activity execution time.
Violation handling	Once a throughput violation is detected, violation handling will be triggered to locate the queuing systems where time delays occurred and add a new server instance to that queueing system to compensate for the time delays.

7 EVALUATION

In this section, we demonstrate and validate the performance of our strategy TVS_{pro} and compare it with other representative strategies. The simulation experiments are conducted in SwinFlow-Cloud.

7.1 Experimental Settings for Workflows

Motivated by the securities exchange business process, we simulate a continuous running of a large number of parallel workflow instances as described in Table 2. We conduct the experiments with 3 different batches of workflows where the number of workflow instances increases from 3000, 6000 to 10000. To simplify the experiments and focus on the evaluation of temporal verification strategy itself, all simulated workflow instances are composed of only sequential activities. Workflow with complex structure (*e.g.,* choice, iteration and parallelism) can be pre-processed using some strategies such as workflow flattening. Business workflows with 15, 20 and 25 sequential activities are tested to evaluate the effect of different workflow sizes on the effectiveness of these strategies. The mean execution time is randomly generated from the range of 5 to 15 seconds. Coefficient of variation is assigned as 0.2 similar to the work in [14] to ensure the consistency of comparison. We use an online queueing calculator[3] provided by Kardi Teknomo to calculate some mean values such as queueing length and waiting time in the queueing systems for cloud services. Thus, the expected response time of each workflow activity can be accurately obtained. Real arrival time and execution time of activities are designed to follow general distribution which are simulated by *Simulink*[4].

Random noises are also generated to simulate large delays along workflow execution due to unpredictable causes such as network congestion or other critical software errors. These delays are often too large to be simulated by random distribution models. We randomly select 5% of the total activities, and increase their execution times by 0%, 25% and 50% according to the noise level.

The deadline assignment strategy proposed in [5] is adopted in this paper where a confidence value of 90% is specified, namely the target on-time completion rate is 90%. The upper bound temporal constraint of the workflow is defined as 120% of the total mean activity response time to simulate a reasonable deadline. Here, 120% threshold is set as a reference based on the *3-sigma rule* in normal distribution. Note that 120% is selected as a reasonable threshold. There may exist an optimal threshold which can help to accurately distinguish the local violation states from non-violation states for workflow activities. We will leave the question about how to obtain the optimal threshold of local temporal violations as one of our future work.

The deadline constraints in the Service Level Agreement specifies that the batch of workflows needs to be completed within 2 hours. The basic time unit is set as equal interval of one minute, so there are 120 candidate checkpoints along the system timeline. The violation handling strategy adopted for these experiments is the one proposed in [15] where a server instance with fixed lifecycle will be added to the selected queueing system to compensate for the occurred time delays. Each round of experiment is repeated for 15 times to get the average experimental results.

In our experiments, we compare our strategy TVS_{pro} with two representative temporal verification strategies. In our previous work [9], throughput based verification strategy has been proved to be more efficient and effective than all existing response-time based temporal verification strategies. Therefore, we only focus on the comparison with a throughput-based strategy. The basic idea of the two representative strategies are described as follows:

- TVS_{thr}: It is a throughput-based temporal verification strategy which does not consider the effect of time delay propagation [16]. It takes every candidate time point as a checkpoint given in Definition 3.
- $TVS_{n\&s}$: It is a throughput-based temporal verification strategy which takes each workflow activity as a candidate checkpoint [8].

To get the baseline results for comparison purpose, we record the on-time completion rates of workflow instances under natural situation, *i,e.* without any temporal verification or violation handling strategies (denoted as **NIL**).

3. http://people.revoledu.com/kardi/tutorial/Queuing/index.html

4: https://www.mathworks.com/products/simulink.html

7.2 Experimental Results and Analysis

Efficiency: The efficiency of temporal verification strategy can be measured by time overhead of each strategy. The overall time overhead is the accumulation of overhead at each candidate checkpoint including both computation and communication overhead.

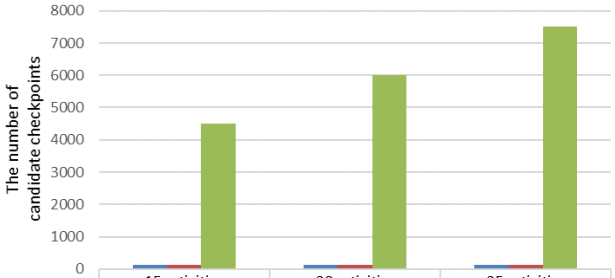
According to the *throughput conformance verification strategy* depicted in Table 1, both checkpoint selection and temporal verification have low computational cost since they only require simple calculations. This is consistent to our experimental results which indicate that the computation overhead of verification strategy is very small (in milliseconds). Given the fact that the durations of workflow activities are normally on the order of seconds or minutes, the computation overhead can be considered negligible.

In contrast, the majority time overhead for temporal verification strategies is the communication time for acquiring runtime information, such as reading system log to obtain the start time and end time of workflow activities. For temporal verification strategies that take time points as checkpoints (*e.g.*, TVS_{pro} and TVS_{thr}), communication is required once at each candidate system time point, each communication needs to read the time-related information of all workflow activities within the basic observation time unit. While for the strategies that take every workflow activities as checkpoint (*e.g.*, $TVS_{n\&s}$), communication is required at every workflow activity, but each communication only needs to read the data of one activity. So the question arises: which communication overhead is larger?

Recently, an analogy study on EC2 has been conducted [16]. The authors have recorded and analyzed the communication time for reading different bytes of date from the same S3 (Simple Storage Service) file. Experimental results demonstrate that the average reading time are very close despite of huge difference in the data size. Therefore, for the above two kinds of strategies, overhead at a candidate checkpoint is almost the same since the communication overhead is close and computation overhead can be neglected. The difference of overall time overhead between the two kinds of verification strategies mainly lies in different number of candidate checkpoints.

Figure 4 depicts the average number of candidate checkpoints by each strategy. The number of parallel workflow instances is 3000. TVS_{pro} and TVS_{thr} take time points as candidate checkpoints. Despite the increase of instance size, there are a constant of 120 candidate checkpoints when the basic time unit is set as equal interval of one minute. In contrast, $TVS_{n\&s}$ selects much more candidate checkpoints than the other two strategies since it is working at each workflow activity. Thus, every workflow activity is regarded as a candidate checkpoint, the number of candidate checkpoints increases with workflow instance size and the number of parallel workflow instances. Compared with $TVS_{n\&s}$, the reduction rate of candidate checkpoints for TVS_{pro} and TVS_{thr} are both 98.4% when workflow instance size reaches 25. The results for other two experiments with 6000 and 10000 parallel workflow instances are similar. Due to the page limit, more details are omitted here.

Effectiveness: On-time completion rate is a critical indicator for temporal QoS of business workflows. While as



	15 activities	20 activities	25 activities
TVSpro	120	120	120
TVSthr	120	120	120
TVSn&s	4500	6000	7500

Fig. 4. Numbers of candidate checkpoints by each strategy

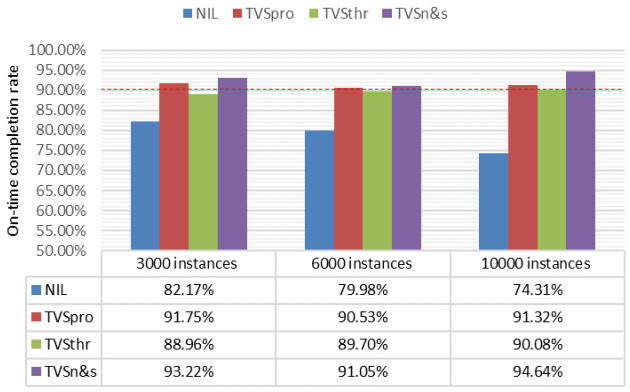
mentioned in the introduction, strictly on-time completion is hardly possible. Higher service quality is also not necessary in business scenarios because the service provider needs to cover the cost of over-provisioned resources. Therefore, an effective strategy is the one that reaches target on-time completion rate (90% in our experiments) with the least amount of resources consumption. In our experiments, when a throughput violation is detected, new resources will be added into queueing systems at handling points to speed up activities execution. Thus, the number of handling points can be regarded as an alternative performance parameter to measure the resource consumption of verification strategies. We also measure the average handling points for every 1% increment from the baseline on-time completion rate (namely the on-time completion rate achieved by **NIL**), which can represent the cost-effectiveness for the resources used for violation handling. The formula is as follows:

$$\frac{H}{\gamma'\% - \gamma\%} \tag{6}$$

Where H denotes the number of total handling points needed by each strategy, $\gamma\%$ is the baseline on-time completion rate and $\gamma'\%$ is the on-time completion rate achieved by each strategy.

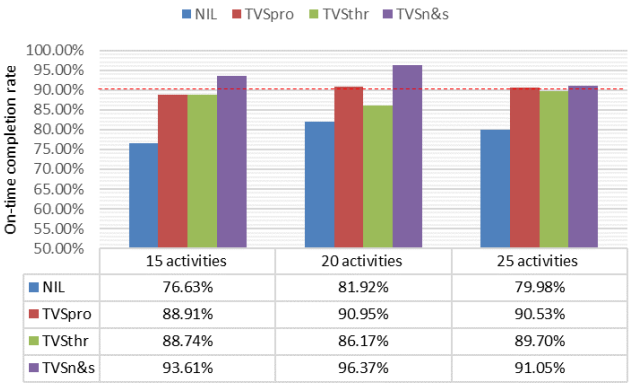
Figure 5 depicts the real on-time completion rates by each strategy with different number of parallel workflow instances when workflow instance size is 25. Other results such as the number of detected throughput violations and handling points are recorded in Table 3.

As shown in Figure 5, each strategy can significantly improve the on-time completion rate when compared with the baseline. $TVS_{n\&s}$ can achieve a higher on-time completion rate than the other two strategies. Both TVS_{pro} and $TVS_{n\&s}$ can achieve target on-time completion rate 90% (the red dashed line in Figure 5). In contrast, TVS_{thr} maintains nearly 90% on-time completion rate but slightly less than the target when the number of workflow instances are 3000 and 6000. However, as shown in Table 3, since $TVS_{n\&s}$ is working at each workflow activity, the number of detected throughput violations and handling points are much more than the other two strategies. The average handling points for every 1% increment from the baseline on-time completion rate by $TVS_{n\&s}$ is several times more than TVS_{pro} and TVS_{thr}, which



	3000 instances	6000 instances	10000 instances
NIL	82.17%	79.98%	74.31%
TVSpro	91.75%	90.53%	91.32%
TVSthr	88.96%	89.70%	90.08%
TVSn&s	93.22%	91.05%	94.64%

Fig. 5. On-time completion rates with different number of instances



	15 activities	20 activities	25 activities
NIL	76.63%	81.92%	79.98%
TVSpro	88.91%	90.95%	90.53%
TVSthr	88.74%	86.17%	89.70%
TVSn&s	93.61%	96.37%	91.05%

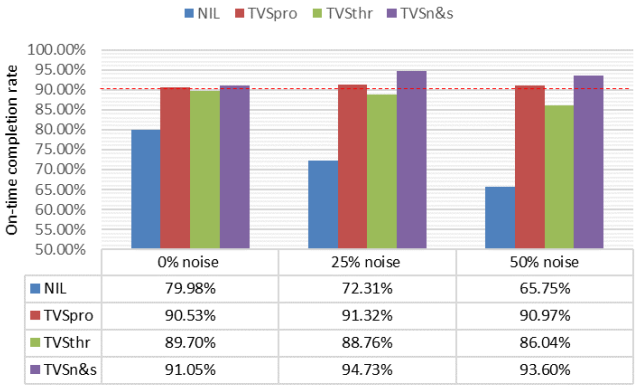
Fig. 6. On-time completion rates with different workflow sizes

TABLE 3
RESULTS WITH DIFFERENT PARALLEL INSTANCES

Strategies	Detected throughput violations			Handling points			Average handling points for every 1% increment from the baseline		
	3000	6000	10000	3000	6000	10000	3000	6000	10000
TVS_{pro}	40	41	81	126	147	367	13.15	13.93	21.57
TVS_{thr}	40	36	75	112	130	351	16.49	13.37	22.76
$TVS_{n\&s}$	532	873	1726	532	873	1726	48.14	78.86	84.90

TABLE 4
RESULTS WITH DIFFERENT WORKFLOW SIZES

Strategies	Detected throughput violations			Handling points			Average handling points for every 1% increment from the baseline		
	15	20	25	15	20	25	15	20	25
TVS_{pro}	63	83	41	171	201	147	13.93	22.26	13.93
TVS_{thr}	63	52	36	171	166	130	14.12	27.29	13.37
$TVS_{n\&s}$	568	702	873	568	702	873	33.45	48.58	78.86



	0% noise	25% noise	50% noise
NIL	79.98%	72.31%	65.75%
TVSpro	90.53%	91.32%	90.97%
TVSthr	89.70%	88.76%	86.04%
TVSn&s	91.05%	94.73%	93.60%

Fig. 7. On-time completion rates with different noise levels

TABLE 5
RESULTS WITH DIFFERENT NOISE LEVELS

Strategies	Detected throughput violations			Handling points			Average handling points for 1% increment from the baseline		
	0%	25%	50%	0%	25%	50%	0%	25%	50%
TVSpro	41	76	88	147	392	413	13.93	20.62	16.38
TVSthr	36	63	71	130	314	335	13.37	19.09	16.51
$TVS_{n\&s}$	873	1128	1895	873	1128	1895	78.86	50.31	68.04

means the required resources for temporal violation handling is dozens of times more than the other two strategies. TVS_{pro} and TVS_{pro} detect roughly the same number of throughput violations and require similar numbers of handling points because both of them take every candidate time point as a checkpoint. Compared with TVS_{thr}, TVS_{pro} can maintain over 90% on-time completion rate. The reason is that TVS_{pro} can detect more violations when taking delay propagation effect into consideration.

Figure 6 and Table 4 show the experimental results under different workflow sizes. The number of parallel workflow instances is 6000. For the same reason, $TVS_{n\&s}$ can achieve the highest on-time completion rate with much more violation handling points and resources consumption. TVS_{thr} fails to achieve the target completion rate in all cases, while our strategy TVS_{pro} can achieve over 90% on-time completion rate when the workflow size is larger than 20. However, when workflow size is 15, the on-time completion rate of TVS_{pro} is less than target rate 90%. We speculate the reason is that the delay propagation effect is less detectable when workflow size becomes smaller. Since TVS_{pro} can detects more violations, the number of handling points is larger when compared with TVS_{thr}. But the performance of TVS_{pro} is still better than $TVS_{n\&s}$ and TVS_{thr}.

Finally, we compared our strategy TVS_{pro} with the other two strategies under different levels of noise, which are shown in Figure 7 and Table 5. The workflow instance size is 25. $TVS_{n\&s}$ unsurprisingly achieves the highest on-time completion rate with much more violation handling points and resource consumption. TVS_{thr} requires the least amount of handling points and new resources, but it fails to meet the target on-time completion rate in all cases. Our strategy TVS_{pro}, by contrast, can ensure steady and satisfactory on-time completion rates under different noise levels. The resource consumption for temporal violation handling is slightly larger than TVS_{thr}, but is dozens of times less than $TVS_{n\&s}$. Compared with TVS_{thr}, the advantage of TVS_{pro} becomes more evident when the level of noise increases. This shows that our strategy can achieve a satisfactory result when the workflow system suffers from unexpected situations where large delays have occurred.

In conclusion, our experimental results demonstrate

that our novel propagation-aware verification strategy can reach the target on-time completion rate of parallel workflow instances with less resource consumption. It is generally better in terms of both in efficiency and effectiveness than the state-of-the-art strategy. In addition, our strategy is scalable and can be applied to fluctuating workload environments such as the cloud.

7.3 Threads to validity

External threats to validity. The main threat to the external validity of our experiments is the representativeness of our motivating business workflow example which sets the background and affects the parameter settings. This securities exchange business process is a typical instance-intensive business process which is used by many researchers to motivate and analyze the needs for business cloud workflows. However, different business processes may have different features. To mitigate this threat, in our experimental settings we deliberately increased the search space by exploring different parameter settings to test the more general applicability of our strategy.

Internal threats to validity. The main threat to the internal validity is the comprehensiveness of our experiments. To mitigate this threat, as is a common practice for simulation-based experiments [17], we adopt two general rules in our experiments: (1) for static parameters such as coefficient of variation, we choose representative settings based on the earlier related work [8], [16]; (2) for dynamic parameters such as workflow durations, we explored the search space to a large degree by predefining a series of candidates or generating randomly on-the-fly.

8 Related Work

Conventional runtime verification approaches. Runtime verification is pursued as a lightweight technology, where a monitor checks at runtime whether or not the execution of a system under scrutiny satisfies a given correctness property [18]. According to the working mechanisms of the monitor, monitoring approaches in runtime verification can be classified as event-triggered or time-triggered. Most literatures on runtime verification focus on event-triggered solutions, in the sense that a monitor is invoked for analysis as soon as any event of interest occurs [19], [20]. However, this frequent invocation induces significant runtime overhead to the system. Some approaches attempt to mitigate this by *e.g.* improving instrumentation [21], combining static and dynamic analysis technologies [22]. However, other inherent limitations, such as unpredictable invocation of the monitor and possible bursts of monitoring invocation, cannot be easily addressed. To overcome the above defects, a time-triggered runtime verification approach has been developed [23]. Here runtime monitor is invoked with a constant frequency and takes samples from the program in order to evaluate the properties [24], [25]. In the context of large-scale business workflow system, hundreds of thousands of monitors are needed for monitoring the parallel workflow instances, and thus the overhead is a serious obstacle.

Workflow verification. Workflow verification is a longstanding area of workflow management research. In general, workflow management systems work at two stages: build-time and runtime. Workflow verification at build-time is concerned with determining, in advance, whether a workflow model exhibits certain desirable behaviors. Since the mid-1990s, many researchers have been working on verification technologies at workflow build-time [26], [27]. The need for formal methods in workflow modeling and verification has been widely recognized, which provides rigorous and mathematical semantics to guarantee that a system will comply with target specifications. Typical formal methods include Petri nets, event algebra, state charts, and temporal logic [1], [2]. However, static verification is not sufficient to tackle compliance problems in a comprehensive way due to runtime issues such as cloud platform performance variance.

Workflow runtime verification monitors the running instances of a process and assesses whether they comply with the business constraints of interest. Dimitra *et al.* present a strategy to check a running program against Linear Temporal Logic (LTL) specification [28]. Davide *et al.* provide a framework for the specification and automatic verification of business process based on a temporal extension of answer set programming (ASP) [29]. Fabrizio *et al.* present a runtime verification based on linear temporal logic and colored automata [30]. However, these approaches cannot directly be applied in workflow temporal verification as it is incredibly expensive to repeat these verification strategies thousands of times when dealing with thousands of parallel workflow instances in real-world, large-scale business workflow application domains.

Workflow temporal verification. Temporal verification is the major approach for delivering satisfactory temporal QoS, which focuses on the time-constrained large-scale workflow systems and applications [45]. Initially, most efforts have been dedicated to the temporal verification of scientific workflow applications. A series of studies on temporal consistency model, including binary-state based, multiple-state based as well as continuous-state based models, have been published gradually [34], [35], aiming at providing a more precise method for workflow temporal verification. To deal with large numbers of parallel business workflows, a throughput consistency model has been first proposed in [5] for setting throughput constraints. The work in [9] presents a runtime throughput based temporal consistency model to monitor parallel business workflows. The work in [36] presents a temporal violation transmission model at workflow build-time stage to estimate the number of temporal violations that may occur at runtime. The prediction result can provide essential reference for temporal violation prevention and handling strategy.

QoS-aweare workflow scheduling. Workflow scheduling is a research hotspot in cloud computing which allocates each workflow task to a relevant cloud service by ordering the execution of various resources to obtain satisfactory QoS requirements. The current state-of-the-art research tackles different scheduling problems in cloud workflow systems by focusing on different QoS optimization constraints. Zhang et al. present an iterative ordinal optimization method to achieve high throughput with lower memory demand [37]. Sahni et al. present a dynamic cost-minimization

deadline constrained heuristic for scheduling scientific applications in a public cloud environment, where both time and cost are considered [38]. Some other scheduling algorithms take multiple QoS parameters such as reliability and energy requirements as constraints to schedule workflow tasks to the resources [39, 40]. However, considering the scheduling efficiency and cost, those strategies for single scientific workflow or multi-workflows with low concurrency are not applicable to the scheduling of instance-intensive business workflows.

Benchmarking workflow management system. The performance of WfMS (workflow management systems) has a significant impact on the quality of service provided by hosted workflow applications. In order to assess and compare existing WfMS and workflow engines with the aim of selecting appropriate host for execution, a multitude of benchmarks have emerged. BenchFlow project tries to design and implement the first benchmark to assess and compare the performance of WfMS that is compliant with Business Process Model and Notation 2.0 standard [41]. Betsy is a BPEL/BPMN engine test system, which implements a comprehensive benchmark for workflow engines [42]. Harrer et al. present a pattern language that captures common solutions to reoccurring problems in the area of workflow engine conformance and performance benchmarking, which help future benchmark authors to design and implement new workflow engine benchmarks [43].

9 Conclusion

A major challenge for business process management and service-oriented systems is how to achieve the target on-time completion rate as an essential non-functional requirement of parallel business workflow instances. Current workflow verification approaches cannot be applied directly in this scenario either due to the performance variation of runtime cloud computing environments or their limitations on efficiency and scalability. In this paper, we introduce an effective temporal conformance verification strategy for a large number of business cloud workflows. Instead of response time, workflow throughput was employed as the measurement to monitor parallel workflow instances. A novel propagation-aware throughput conformance verification strategy that considers the effect of delay propagation in cloud system was presented and implemented in a prototype cloud workflow system. Experimental results showed that our strategy outperforms the state-of-the-art strategy in achieving better efficiency and effectiveness.

Acknowledgement

The authors would like to acknowledge the support provided by the grands of the National Natural Science Foundation of China (grant No.61572374, 61300042, U163620068, U1135005). We would like to thank Dr. Dahai Cao for his contribution to the development of SwinFlow-Cloud. Jin Liu and Xiao Liu are corresponding authors.

References

[1] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21-66, 1998.

[2] M. P. Singh, G. Meredith, C. Tomlinson, and P. C. Attie, "An event algebra for specifying and scheduling workflows," *Proc. 4th IEEE Int. Conf. Database Systems for Advanced Applications*, pp. 53-60, 1995.

[3] Y. Falcone, K. Havelund, and G. Reger, "A Tutorial on runtime verification," *J. Engineering Dependable Software Systems*, vol. 34, IOS press, pp. 141-175. 2013,

[4] X. Liu, Y. Yang, D. Cao, and D. Yuan, "Selecting checkpoints along the time line: A novel temporal checkpoint selection strategy for monitoring a batch of parallel business processes," *Proc. 35th ACM/IEEE Int. Conf. Software Engineering*, pp. 1281-1284, 2013.

[5] X. Liu, D. Wang, D. Yuan, and Y. Yang, "A novel deadline assignment strategy for a large batch of parallel tasks with soft deadlines in the cloud," *Proc. 15th IEEE Int. Conf. High Performance Computing and Communications & 10th IEEE Int. Conf. Embedded and Ubiquitous Computing (HPCC_EUC)*, pp. 51-58, 2013.

[6] X. Liu, Y. Yang, D. Yuan, and J. Chen, "Do we need to handle every temporal violation in scientific workflow systems?" *ACM Trans Software Engineering and Methodology (TOSEM)*, vol. 23, no. 1, pp. 1-34, 2014.

[7] G. J. Nutt "Tutorial: computer system monitors," *ACM SIGMETRICS Performance Evaluation Review*, vol. 5, no. 1, pp. 41-51, 1976.

[8] F. Wang, X. Liu, and Y. Yang, "Necessary and sufficient checkpoint selection for temporal verification of high-confidence cloud workflow systems," *Science China Information Sciences*, vol. 58, no. 5, pp. 1-16, 2015.

[9] X. Liu, D. Wang, D. Yuan, F. Wang and Y. Yang, "Throughput based temporal verification for monitoring large batch of parallel processes," *Proc. 2014 ACM Int. Conf. Software & System Process*, pp. 124-133, 2014

[10] X. Chang, B. Wang, J. Muppala, and J. Liu, "Modeling active virtual machines on IaaS Clouds using an M/G/m/m+K queue," *IEEE Trans. Services Computing*, vol. 9, no. 3, pp. 408-420, 2016.

[11] T. Atmaca, T. Begin, A. Brandwajn, and H. Castel, "Reducing the complexity of the performance analysis of a multi-server facilities," Research Report RR-8617, Institut Telecom, Telecom SudParis, Evry, France; Université Lyon 1/LIP (UMR INRIA, ENS Lyon CNRS, UCBL), Lyon, France; University of California Santa Cruz, Baskin School of Engineering, USA, 2014.

[12] X. Liu, Z. Ni, D. Yuan, et al., "A novel statistical time-series pattern based interval forecasting strategy for activity durations in workflow systems," *J. Systems and Software*, vol. 84, no. 3, pp. 354-376, 2011.

[13] X. Liu, J. Chen, Z. Wu, Z. Ni, D. Yuan, and Y. Yang, "Handling recoverable temporal violations in scientific workflow systems: a workflow rescheduling based strategy," *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing*, pp. 534-537, 2010.

[14] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no.5, pp. 936-943, 2012.

[15] H. Luo, X. Liu, J. Liu and F. Wang, "Where to fix temporal violations: a novel handling point selection strategy for business cloud workflows," *Proc. IEEE Int. Conf. Services Computing*, pp. 155-162, 2016.

[16] X. Liu, D. Wang, D. Yuan, F. Wang, and Y. Yang, "Workflow temporal verification for monitoring parallel business processes," *J. Software Evolution & Process*, vol. 28, no. 4, pp. 286-302, 2016.

[17] A. M. Law, and W. D. Kelton, *Simulation Modeling and Analysis*, New York: McGraw-Hill, 1991.

[18] M. Leucker, and C. Schallhart. "A brief account of runtime verification,". *J. Logic & Algebraic Programming*, vol. 78, no. 5, pp. 293-303, 2009.

[19] M. Jaber, T. H. Nguyen, M. Bozga, and S. Bensalem, "Runtime verification of component-based systems," *Proc. Int. Conf. Software Engineering and Formal Methods*, Springer Berlin Heidelberg, pp. 204-220, 2011.

[20] Q. Luo, Y. Zhang, C. Lee, et al. "RV-Monitor: Efficient parametric runtime verification with simultaneous properties," *Proc. Int. Conf. Runtime Verification*, Springer International Publishing, pp. 285-300, 2014.

[21] J. Seyster, K. Dixit, X. Huang, et al., "Aspect-oriented instrumentation with GCC," *Proc. Int. Conf. Runtime Verification*, Springer Berlin Heidelberg, pp. 405-420, 2010.

[22] E. Bodden, "Efficient hybrid typestate analysis by determining continuation-equivalent states," *Proc. 32nd ACM/IEEE Int. Conf. Software Engineering*, pp. 5-14, 2010.

[23] B. Bonakdarpour, S. Navabpour, and S. Fischmeister, "Time-triggered runtime verification," *Formal Methods in System Design*, vol. 43, no. 1, pp. 29-60, 2013.

[24] S. Navabpour, B. Bonakdarpour, and S. Fischmeister, "Path-aware time-triggered runtime verification," *Proc. Int. Conf. Runtime Verification*, pp. 199-213, 2012.

[25] S. Navabpour, Y. Joshi, W. Wu, et al., "RiTHM: A tool for enabling time-triggered runtime verification for c programs," *Proc. 9th ACM Joint Meeting on Foundations of Software Engineering*, pp. 603-606, 2013.

[26] C. E. Gerede, and J. Su, "Specification and verification of artifact behaviors in business process models," *Proc. Int. Conf. Service-Oriented Computing*, Springer Berlin Heidelberg, pp. 181-192, 2007.

[27] D. Edmond, M. T. Wynn, A.H.M.T. Hofstede, W. M. P. V.D. Aalst, and H. M. W. Verbeek, "Business process verification–finally a reality!" *Business Process Management Journal*, vol. 15, no. 1, pp. 74-92, 2009.

[28] D. Giannakopoulou, and K. Havelund, "Automata-based verification of temporal properties on running programs," *Proc. 16th IEEE Int. Conf. Automated Software Engineering*, pp. 412-416, 2001.

[29] D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. T. Dupré, "Verifying compliance of business processes with temporal answer sets," CILC, pp. 147-161, 2011.

[30] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. V.D. Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," *Proc. Int. Conf. Business Process Management*, Springer Berlin Heidelberg, pp.132-147, 2011.

[31] X. Liu, J. Chen, and Y. Yang, "A probabilistic strategy for setting temporal constraints in scientific workflows," *Proc. Int. Conf. Business Process Management*, Springer Berlin Heidelberg, pp. 180-195, 2008.

[32] J. Chen, and Y. Yang, "Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in grid workflow systems," *ACM Trans. Autonomous & Adaptive Systems*, vol. 2, no. 2, article 6, 2007.

[33] X. Liu, Y. Yang, Y. Jiang, and J. Chen, "Preventing temporal violations in scientific workflows: where and how," *IEEE Trans. Software Engineering*, vol. 37, no. 6, pp. 805-825, 2011.

[34] J. Eder, E. Panagos, and M. Rabinovich, "Time constraints in workflow systems," *Proc 11th Int. Conf. Advanced Information Systems Engineering (CAiSE99)*, pp. 286–300, 1999.

[35] B. S. Lerner, S. Christov, L. J. Osterweil, R. Bendraou, U. Kannengiesser, and A. Wise, "Exception handling patterns for process modelling," *IEEE Trans. Software Engineering*, vol. 36, no. 2, pp. 162-18, 2010.

[36] H. Luo, J. Liu, X. Liu and Y. Yang, "Predicting temporal violations for parallel business cloud workflows", *Software: Practice and Experience*, vol. 48, no. 4, pp. 775-795, 2018.

[37] F. Zhang, J. Cao, K. Hwang, et al, "Adaptive workflow scheduling on cloud computing platforms with iterative ordinal optimization", *IEEE Trans. Cloud Computing*, vol. 3, no. 2, pp. 156-168, 2015.

[38] J. Sahni, D. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment", *IEEE Trans. Cloud Computing*, vol. 6, no. 1, pp. 2-18, 2018.

[39] Z. Li, J. Ge, H. Hu, et al., "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds", *IEEE Trans. Services Computing*, vol. 11, no. 4, pp. 713-726, 2018.

[40] L. Zhao, Y. Ren, K. Sakurai, "Reliable workflow scheduling with less resource redundancy", Parallel Computing, vol. 39, nol. 10, pp. 567-585, 2013.

[41] V. Ferme, A. Ivanchikj, C, Pautasso, "A framework for benchmarking BPMN 2.0 workflow management systems", *Proc. Int. Conf. Business Process Management*, Springer, Cham, pp. 251-259, 2016.

[42] M. Geiger, S. Harrer, J. Lenhard, "Process engine benchmarking with Betsy-Current status and future directions", ZEUS, pp. 37-44, 2016.

[43] S. Harrer, J. Lenhard, O. Kopp, et al., "A Pattern Language for Workflow Engine Conformance and Performance Benchmarking", *Proc. 22nd European Conference on Pattern Languages of Programs*, pp. 1-46, 2017.

[44] D. Cao, X. Liu and Y. Yang," Novel client-cloud architecture for scalable instance-intensive workflow systems", *Proc. 14th Int. Conf. Web Information System Engineering*, pp. 270-284, 2013.

[45] X. Liu, J. Chen, and Y. Yang, *Temporal QoS management in scientific cloud workflow systems*, Elsevier, 2012.

Haoyu Luo received his PhD. degree from Wuhan University, China, in 2018. He received his Bachelor degree from East China Institute of Technology in 2011 and Master degree from Northwest Normal University in 2014. He is currently an Associate Researcher in School of Computer Science, South China Normal University. His research interests include workflow system and cloud computing.

Xiao Liu received his PhD degree from t Swinburne University of Technology, Australia, in 2011. He received his Master and Bachelor degree from Hefei University of Technology, China, in 2007 and 2004 respectively. He is currently a Senior Lecturer at School of Information Technology, Deakin University, Australia. His research interests include workflow system and cloud computing. More details about his research can be found at: https://sites.google.com/site/drxiaoliu/

Jin Liu received his PhD degree from Wuhan University, China, in 2005. He is now a full professor in School of Computer Science, Wuhan University. His current research interests include software service engineering and software repository mining. He has published more than 70 papers in well-known conferences and journals.

Yun Yang received the PhD degree from the University of Queensland, Australia, in 1992. He is a full professor with Swinburne University of Technology. His research interests include software engineering, cloud computing, workflow systems and service-oriented computing. More details about his research can be found at https://www.swinburne.edu.au/science-engineering-technology/staff/profile/index.php?id=yyang.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently a Professor of Software Engineering at Monash University, Australia. He is a Fellow of Automated Software Engineering, Fellow of Engineers Australia. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.