Workflow temporal verification for monitoring parallel business processes

Xiao Liu1,2,*, Dingxian Wang1, Dong Yuan3, Futian Wang4 and Yun Yang5

1Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
2School of Information Technology, Deakin University, Melbourne, Australia
3School of Electrical and Information Engineering, The University of Sydney, Sydney, Australia
4School of Computer Science and Technology, Anhui University, Anhui, China
5School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia

ABSTRACT

Workfl ow temporal verification is conducted to guarantee on-time completion, which is one of the most important QoS (Quality of Service) dimensions for business processes running in the cloud. However, as today’s business systems often need to handle a large number of concurrent customer requests, conventional response-time based process monitoring strategies conducted in a one-by-one fashion cannot be applied efficiently to a large batch of parallel processes because of significant time overhead. Similar situations may also exist in software companies where multiple software projects are carried out at the same time by software developers. To address such a problem, based on a novel runtime throughput consistency model, this paper proposes a QoS-aware throughput based checkpoint selection strategy, which can dynamically select a small number of checkpoints along the system timeline to facilitate the temporal verification of throughput constraints and achieve the target on-time completion rate. Experimental results demonstrate that our strategy can achieve the best efﬁciency and effectiveness compared with the state-of-the-art as and other representative response-time based checkpoint selection strategies. Copyright © 2016 John Wiley & Sons, Ltd.

Received 7 January 2015; Revised 12 July 2015; Accepted 23 October 2015

KEY WORDS: temporal verification; checkpoint selection; parallel processes; quality of service; cloud computing

1. INTRODUCTION

The rapid growth of e-government and e-business demands fast and cost-effective processing of a large number of customer requests in a constrained period of time. For example, a government taxation ofﬁce needs to process hundreds of thousands of tax declarations every day during the peak period, and the tax return process for each client request may need to be completed within 2 weeks [1, 2]. Failures of completing these processes in time will result in signiﬁcant deterioration of customer satisfaction and even huge ﬁnancial losses. Therefore, on-time completion becomes one of the most important QoS dimensions that pervade the design, development, and running of business process management systems, for example, the cloud workﬂow system [2]. In the meantime, cloud computing is establishing itself as a new paradigm for delivering information technology (IT) infrastructure elements such as computing, storage and

*Correspondence to: Xiao Liu, School of Information Technology, Deakin University, Melbourne, Australia.
†E-mail: xiao.liu@deakin.edu.au
‡The initial work was published in the proceedings of 2014 International Conference on Software and System Process, pp.124–133, Nanjing, China, May 26–28, 2014.
network resources as IT services over the Internet [3]. Customers can access these services in a pay-as-you-go fashion while avoiding huge capital investment, energy consumption, and system maintenance. Cloud computing can offer on-demand, elastic, cost-effective hardware and software resources, which is an ideal hosting environment for running of a large number of parallel business processes [4–7]. However, because of its dynamic nature, to guarantee the delivery of satisfactory service quality is a big challenge. There are many efforts from both Software Engineering [8, 9] and Parallel and Distributed Computing areas [10] dedicated to the quality assurance of cloud and general Web services [11].

In the meantime, software process can also be regarded as a kind of business process. Similar situations also exist in software companies where multiple concurrent software projects with specific deadlines are carried out by software developers. There are many methodologies and tools available on the market to assist project managers in such situations. For example, over 100 tools including 2-plan, Trello, and Microsoft Team Foundation are listed on the Wikipedia page about the comparison of project management software. Although many of them have the features of project scheduling, resource management and some of them even have workflow and time tracking features, it is still a big challenge to effectively monitor the progress of each project and predict whether they can be delivered in time. Considering their similarities, the studies for the management of parallel business processes can also be applied to multiple software projects. However, the most significant difference between a software process and a general business process (specifically cloud workflow in this paper) is that the underlying ‘processing unit’ is people instead of cloud services (e.g. virtual machines). Therefore, we need to change the underlying resource model from cloud service based to developer based if we want to apply the ideas of this paper to the monitoring of software processes. However, as this paper focuses on the monitoring of cloud business processes, we will leave the issue for software process as one of the future research directions.

Workflow temporal verification is the major approach for delivering satisfactory temporal QoS in workflow systems. Given a typical workflow lifecycle, a general temporal verification framework may contain four basic components, viz. temporal lifecycle, temporal constraint setting, temporal-aware service selection, temporal consistency monitoring, and temporal violation handling [12]. Among them, temporal consistency monitoring consists of temporal checkpoint selection and temporal verification. Temporal checkpoint selection selects a subset of workflow activities as checkpoints for the verification of temporal consistency states, and temporal verification measures the current temporal consistency state at the checkpoint and reports whether a temporal violation occurs or not [13]. Here ‘a temporal violation’ means an intermediate violation, which can be fixed locally to ensure the overall timely completion. When a temporal violation is detected, temporal violation handling strategies such as workflow rescheduling or adding new resources will be triggered to compensate for the time delays [14].

In this paper, we focus on temporal consistency monitoring for a large batch of parallel business processes running in a cloud computing environment. More specifically, we focus on those business processes with computational tasks where the underlying processing units are virtual machines. Considering many related studies are in the workflow area, term ‘business process’ is interchangeable with term ‘workflow’ in this paper. In recent years, there are a lot of efforts dedicated to the temporal verification of single large-scale distributed scientific workflow applications in the grid or cloud computing environments [15]. However, to deal with business processes, current strategies for scientific workflows may become much less useful because of their significant differences [16]. Specifically, there are three big challenges as follows.

Challenge #1: Single process versus a large batch of parallel processes: A scientific workflow is a single process with hundreds of thousands of data and computation intensive activities running for hours or even days. In contrast, business processes are often parallel processes for a large number of concurrent customer requests; each may only have tens of activities running in seconds or minutes. Repeating the strategies for monitoring a single process to deal with a large batch of parallel processes may be an intuitive solution, but will inevitably introduce a large amount of time and cost overheads. Therefore, a big challenge for monitoring business processes is how to efficiently measure the temporal consistency states of a large batch of parallel processes.

 Challenge 1: Single process versus a large batch of parallel processes: A scientific workflow is a single process with hundreds of thousands of data and computation intensive activities running for hours or even days. In contrast, business processes are often parallel processes for a large number of concurrent customer requests; each may only have tens of activities running in seconds or minutes. Repeating the strategies for monitoring a single process to deal with a large batch of parallel processes may be an intuitive solution, but will inevitably introduce a large amount of time and cost overheads. Therefore, a big challenge for monitoring business processes is how to efficiently measure the temporal consistency states of a large batch of parallel processes.

Challenge #2: Dedicated and static versus shared and dynamic resource environments: Scientific workflows are normally carried out by high performance IT infrastructures such as community-based clusters and grids where resources are usually reserved in advance and dedicated during reservation. Therefore, the performance of the underlying resources is relatively static, and activity durations can normally be estimated by simple statistic models [1]. In contrast, with the emergence of cloud computing, business processes would run in a shared and dynamic public commercial infrastructure with virtually unlimited resources to satisfy the ever increasing need of customers. In the cloud, resources can be elastically provisioned according to the real-time system requirements. However, the number of concurrent customer requests is often hard to predict in a real-world business market where spikes often appear. Therefore, we need to apply more advanced prediction models for cloud services.

Challenge #3: Best-effort versus strict service quality constraints: Scientific workflows are usually running in community-based clusters and grids where best-effort based strategies are adopted. In contrast, business processes running in the cloud have strict constraints on time, cost, reliability, and other QoS requirements because customers are paying for different prices for different levels of service quality. If failed in delivering the promised service quality, the service provider will have to compensate the customer according to the service contract [3]. Meanwhile, another situation that the service provider wants to diminish is over-provisioning, which means the delivered service quality is higher than what the customer pays for. In such a case, the extra cost will be covered by the service provider. Therefore, a big challenge is how to achieve the target on-time completion rate for business processes, but without over-provisioning cloud resources [17].

Given the aforementioned three big challenges, conventional temporal consistency monitoring strategies for single large-scale scientific workflows cannot be applied directly to business processes. To address this problem, in this paper, we investigate the measurement of throughput instead of response time for monitoring a large batch of parallel processes. Based on a novel runtime throughput consistency model, we propose a QoS-aware throughput based checkpoint selection strategy that can dynamically choose a small number of checkpoints along the system timeline to facilitate the temporal verification of throughput constraints and guarantee the target on-time completion rate. Comprehensive experimental results demonstrate that our QoS-aware throughput based checkpoint selection strategy can achieve the most significant reduction in the number of checkpoints while achieving the best closeness to the target service quality compared with the state-of-the-art and other representative response-time based checkpoint selection strategies. In other words, it achieves the best efficiency and effectiveness.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents some preliminary definitions for workflow throughput and throughput checkpoints. Section 4 proposes the novel runtime throughput consistency model, and then, Section 5 proposes the novel throughput consistency verification strategy. Section 6 demonstrates the comprehensive experimental results. Finally, Section 7 concludes the paper and points out some future work.

2. RELATED WORK

Time-related issues cover a wide spectrum of topics in software engineering such as specification, design, verification, testing, exception handling, and software process management [2, 3, 12, 18–21]. In recent years, with the emergency of market-oriented distributed computing infrastructures such as utility-based grid and cloud, time as one of the most important QoS dimensions has attracted increasing interests from both researchers and practitioners in the area of software engineering for distributed systems [9, 10]. Compared with a single service, the temporal QoS of a process, which requires the joint effort of a collection of services, is much more general and complicated.

A general workflow temporal QoS framework was proposed in [12], which can deliver a lifecycle QoS support. In recent years, many checkpoint selection strategies, from intuitive rule based to sophisticated model based, have been proposed. The work in [22] takes every workflow activity as a checkpoint. The work in [23] selects the start activity as a checkpoint and adds a new checkpoint after
each decision activity is executed. It also mentions a type of static activity point which is defined by users at the build-time stage. There are some other strategies such as the one that selects an activity as a checkpoint if its execution time exceeds the maximum duration and the one that selects an activity as a checkpoint if its execution time exceeds the mean duration [24, 25]. The checkpoint selection, which satisfies the property of necessity and sufficiency, is proposed in [26] where an activity point is selected as a checkpoint if and only if its execution time is larger than the sum of its mean duration and its minimum time redundancy. Based on that, the work in [13] further improves the efficiency of temporal verification by utilizing the temporal dependency between temporal constraints.

As for temporal verification, its effectiveness mainly depends on the employed temporal consistency model. A temporal consistency model defines the relationship between the current workflow execution state and the target deadline. A binary-state based temporal consistency model only defines consistency state and inconsistency state, which provides very limited information for the system to judge how serious the inconsistency state is. To overcome this problem, a multiple-state based temporal consistency model is proposed so that different levels of temporal violations can be tackled by different exception-handling strategies [27–30]. Recently, a continuous-state based temporal consistency model is proposed, which can measure the temporal consistency state using confidence values such as 90% [1]. Based on such a model, temporal violations can be handled in a very fine-grained level, and thus, many subtle situations such as ‘auto-recovery’ (namely, the current temporal violation can be automatically recovered by the time redundancy of subsequent activities) is discovered to further reduce the handling cost [31].

The state-of-the-art response-time based checkpoint selection and temporal verification strategies have been proved to be very successful in monitoring single, complex, and large-size scientific workflows. However, there is a so far very limited work that investigates the monitoring of a large batch of parallel business processes. Our latest work in [5] proposed a novel idea of throughput-based equal-distribution checkpoint selection and demonstrated some preliminary results. In this paper, we will further investigate this idea and propose the complete models and new methods for temporal consistency monitoring of parallel processes.

3. PRELIMINARY

In general, response time and throughput are the two most popular performance measurements [6]. Workflow throughput, namely the throughput of a workflow system, is the number of workflows that have been completed in a basic observation time unit [18]. Workflow throughput is a better measurement to address Challenge #1 as mentioned in Section 1. However, such a definition is often too coarse-grained for system monitoring and control. Therefore, in many studies, workflow throughput can also be measured by the number of workflow activities that have been completed in a basic observation time unit [32]. For example, if one workflow activity running for 2 min and another one running for 20 s are both completed during the same observation time unit, their contributions to the system throughput are treated the same, that is, both accounted for one activity completion. However, their actual contributions for meeting the deadlines are very different. To address such a problem, a novel workflow throughput definition, which considers the difference of activity durations, is proposed in our recent work [5].

In addition, we also need to consider different types of temporal constraints. Generally speaking, temporal constraints mainly include three types: upper bound, lower bound, and fixed time [27]. An upper-bound constraint between two activities is a relative-time value so that the duration between them must be less than or equal to it. A lower-bound constraint between two activities is a relative-time value so that the duration between them must be greater or equal to it. A fixed-time constraint at an activity is an absolute-time value by which the activity must be completed. As discussed in [26], conceptually, a lower-bound constraint is symmetrical to an upper-bound constraint, and a fixed-time constraint can be regarded as a special case of an upper-bound constraint of which the start time of the workflow is determined. Hence, upper bound is the most general type of temporal constraints. Therefore, in this paper, we only focus on upper-bound constraints. Our previous work on fixed-time constraints can be found in [33].
Here, we define some basic annotations: $a_i$ is a workflow activity with its runtime (i.e., real), expected, mean, minimum, and maximum durations denoted as $R(a_i)$, $E(a_i)$, $M(a_i)$, $d(a_i)$, and $D(a_i)$, respectively; the activity duration weight of $a_i$ is denoted as $w_i$, which represents the influence of the process structure such as sequence, parallelism, iteration, and choice to the completion time of the entire workflow [1]: $WF_i$ is a workflow with its runtime, expected, mean, minimum, and maximum completion time denoted as $E(WF_i)$, $M(WF_i)$, $d(WF_i)$, and $D(WF_i)$, respectively; $U(WF_i)$ denotes an upper-bound constraint, which means $WF_i$ needs to be completed within such a time period. The basic observation time unit (i.e., the interval for two consecutive throughput measurement) is denoted as $bt$.

**Definition 1**
(Workflow throughput). Given a batch of $q$ parallel workflows Batch{$WF_1, WF_2, ..., WF_q$}, which starts at system time $S_0$, the completion of a workflow activity $a_i$ contributes to the completion of the entire batch of workflows with a value of $w_iM(a_i)/T$ where $T = \sum_{i=1}^{q} M(WF_i)$. Here, assume at the current observation time point $S_t$, the set of new completed activities from the last nearest observation time point $S_{t-1}$ (i.e., $S_t - S_{t-1} = bt$) is denoted as $a_i\{S_{t-1}\}$, then the current system throughput is defined as $TH_{S_{t-1}}^{S_t} = W \times M(a_i\{S_{t-1}\})/T$ where $W$ is an array of activity duration weight $w_i$ representing the influence of the process structures.

Please be noted that for upper bound constraints, $S_0$ is normally assumed to be zero, which means the business process starts at system time zero. However, to be consistent with our previous work, we still use $S_0$ for presentation. Given this new definition, we can clearly measure how much those activities completed during a basic observation time unit contributes to the completion of the entire batch. Furthermore, to facilitate monitoring, we need to assign temporal constraints. Given Definition 1, throughput constraints are the expected accumulated workflow throughputs that should be achieved at a specific system time point. Here, we present an example throughput constraint setting strategy similar as in [17].

**Definition 2**
(An example throughput workflow constraint setting strategy). Given the same batch of workflows as defined in Definition 1 and their upper-bound constraint $U(WF_i)$, at a system time point $S_t$ where $S_t-S_0 = n \times bt$ ($n = 1, 2, 3, ..., \frac{U(WF_i)}{bt}-1$), the throughput constraint assigned at $S_t$ by a throughput constraint setting strategy is $Cons(S_t) = TH_{S_0}^{S_t} = W \times M(a_i\{S_0\})/T$, which means that the expected accumulated system throughput $\sum_{i=1}^{t} TH_{S_0}^{S_t}$ from $S_0$ to $S_t$ should be no less than the value of the assigned throughput constraint.

It can be easily seen that the example throughput constraint setting strategy is to assign the expected percentage of completion to the current system time point. In addition, as there is practically no limit on the position of a constraint point when the basic time unit for monitoring $bt$ is small enough, our strategy can efficiently assign throughput constraints as many as required along the system timeline.

In theory, any time point along the system timeline can be selected as a throughput checkpoint. But in practice, because there is normally a basic time unit for system monitoring, that is, $bt$, throughput checkpoints should be selected accordingly. In this paper, we name the candidate time points for throughput verification as candidate throughput checkpoints. The formal definition is presented as follows.

**Definition 3**
(Candidate throughput checkpoints). Given the same batch of workflows in Definition 1, a system time point $S_t$ along the workflow execution timeline is a candidate throughput checkpoint if $S_t-S_0 = k \times bt$ ($k = 1, 2, 3, ..., \frac{U(WF_i)}{bt}-1$).
As preliminarily discussed in [5], the core idea of throughput-based checkpoint selection is to select system time points for monitoring a batch of parallel processes, which is significantly different from existing response-time based checkpoint selection where activity points are selected for monitoring a single process. Here, the size of the ‘batch’ is not a fixed value but rather determined by the system at runtime. The batch can start at arbitrary time point as long as those parallel processes are having the same or similar deadline. This condition ensures that a common global deadline exists so that a checkpoint selection and temporal verification strategy can be applied simultaneously to all the parallel processes.

4. THROUGHPUT CONSISTENCY MODEL

Temporal verification requires a temporal consistency model. A temporal consistency model is defined for evaluating whether (and to what extent) the target temporal constraints can be satisfied or not [1]. To define a throughput consistency model for runtime monitoring, we need to measure how much throughput has been completed by the current checkpoint and also estimate how much throughput can be completed given the remaining time after the checkpoint. Based on Definition 1, the completed throughput can be easily obtained by checking how many workflow activities have been completed. As for the estimated throughput, it needs to be decided by the estimated running time of the subsequent activities and their remaining time before the deadline. Among many others, statistical models are the most popular for the estimation of activity durations. For example, in the state-of-the-art continuous-state based temporal consistency model, all the activity durations are assumed to follow the normal distribution model $N(\mu_i, \sigma_i^2)$ where $\mu_i$ is the expected value and $\sigma_i$ is the standard deviation. Therefore, according to the ‘3σ rule’, we can assume that $D(a_i) = \mu_i + 3\sigma_i$ and $d(a_i) = \mu_i - 3\sigma_i$.

However, conventional random distribution models are usually employed in the scenarios where the system performance is relatively static and the activity durations are independent to each other. In this paper, as analyzed in Challenge #2, we investigate the cloud computing environment where the underlying services are shared and provisioned elastically according to the number of parallel processes. Therefore, instead of conventional random distribution models, the runtime expected duration $E(a_i)$, that is, $\mu_i$, is estimated using a queuing model, which is a latest work on the performance analysis of cloud computing services [34]. A queuing model is much more powerful and capable of easily adapting to the changes of input tasks and number of provisioned services. To focus on the throughput consistency model, we omit the detailed discussion for the rationale of the model design and present the queuing model used in this paper as follows.

Definition 4

(M/G/m/m+r queuing model for cloud services). In a specific batch of workflows, for $n$ workflow activities of the same type, there are $m$ dedicated services where $n$ is normally much larger than $m$. The queuing model that we adopted is M/G/m/m+r which indicates that the inter-arrival time of requests is exponentially distributed, while task service times are independent and identically distributed random variables that follow a general distribution with mean value of $\mu_i$ for $a_i$. It contains $m$ services and the service order is first come, first serve. The capacity of the system is $m+r$, which means that the buffer size for incoming request is equal to $r$, that is, $n - m$ in this case.

Based on such a queuing model, we can efficiently obtain more accurate expected duration $E(a_i)$, which reflects the dynamic system changes. Please refer to [34] for the formulas of calculating the expected durations, and be noted that, in this paper, an activity duration is the execution time plus the waiting time. There are also many tools available to facilitate the calculation such as popular QtsPlus [35]. The workflow throughput for the remaining time can be defined as follows.

Definition 5

(Estimated workflow throughput). Given the same batch of $q$ parallel workflows in Definition 1, its upper-bound constraint denoted as $U(WF_i)$, at a throughput checkpoint $S_p$, the maximum, expected,
and minimum workflow throughputs for the remaining time are defined with the following formulas:

\[
\begin{align*}
\text{Max} \left(TH \bigg|_{S_p}^{U(WF_i)} \right) &= \left. TH \bigg|_{S_p}^{U(WF_i)} \right) \frac{q \times U(WF_i)}{W \times d \left( a \{ \} \bigg|_{S_p}^{U(WF_i)} \right)} \\
\text{Exp} \left(TH \bigg|_{S_p}^{U(WF_i)} \right) &= \left. TH \bigg|_{S_p}^{U(WF_i)} \right) \frac{q \times U(WF_i)}{W \times E \left( a \{ \} \bigg|_{S_p}^{U(WF_i)} \right)} \\
\text{Min} \left(TH \bigg|_{S_p}^{U(WF_i)} \right) &= \left. TH \bigg|_{S_p}^{U(WF_i)} \right) \frac{q \times U(WF_i)}{W \times D \left( a \{ \} \bigg|_{S_p}^{U(WF_i)} \right)}
\end{align*}
\]

Clearly, the maximum, expected, and minimum workflow throughputs are estimated where the subsequent workflow activities are running with their minimum, expected, and maximum durations, respectively. Based on Definition 5, our novel runtime throughput consistency model is proposed as follows.

**Definition 6**

(Runtime throughput consistency model). Given the same batch of workflows in Definition 1 and its upper-bound constraint \(U(WF_i)\), at a checkpoint \(S_p\), it is said to be of:

1) Absolute Consistency (AC), if

\[
TH \bigg|_{S_p}^{S_0} + \text{Min} \left(TH \bigg|_{S_p}^{U(WF_i)} \right) \geq 100%;
\]

2) Absolute Inconsistency (AI), if

\[
TH \bigg|_{S_p}^{S_0} + \text{Max} \left(TH \bigg|_{S_p}^{U(WF_i)} \right) \leq 100%;
\]

3) \(\alpha\)% Consistency (\(\alpha\)% C), if

\[
F(\lambda_\alpha) = TH \bigg|_{S_p}^{S_0} + \text{Exp} \left(TH \bigg|_{S_p}^{U(WF_i)} \right)
\]

where \(\lambda_\alpha \ (-3 \leq \lambda_\alpha \leq 3)\) is defined as the \(\alpha\)% (0 < \(\alpha\) < 100%) confidence percentile with the cumulative standard normal distribution function of \(F(\lambda_\alpha) = \frac{1}{\sqrt{2\pi}} \int_{-\lambda_\alpha}^{\lambda_\alpha} e^{-x^2/2} dx = \alpha\%\).

According to the ‘3\(\sigma\) rule’, AC (i.e., \(\alpha\% \geq 99.87\%\)) denotes the state that even when every activity is running with its maximum duration (namely, with the minimum workflow throughput), the final deadline can still be met. Therefore, no action for violation handling is required. In contrast, AI (i.e., \(\alpha\% < 0.13\%\)) denotes that even when every activity is running with its minimum duration (namely, with the maximum workflow throughput), the final deadline still cannot be met. Therefore, heavy-weight violation handling strategies are required. Clearly, both AC and AI are two extreme situations, while the rest can be represented by \(\alpha\%\) C, which denotes a probability confidence for on-time completion. This is a better measurement for describing the current service quality. For example, many commercial cloud service providers such as Amazon Web Service uses percentage values like 99%, 99.9%, and 99.99% for service quality on reliability and availability (http://aws.amazon.com/en/s3-sla/). In this paper, we denote such a target service quality as \(\beta\%\) and use 90% as the benchmark, same as in the previous studies [15, 31]. Clearly, when \(\alpha\% \geq \beta\%\), the service quality holds, and no action is required. When \(\alpha\% < \beta\%\), temporal violation handling will be triggered as the current service quality that is below the target. Please refer to [14] for more details on workflow temporal violation handling.
5. NOVEL THROUGHPUT CONSISTENCY VERIFICATION STRATEGY

Based on the novel throughput consistency model presented in Section 4, we propose a novel throughput consistency verification strategy that consists of a throughput-based checkpoint selection strategy, followed by the temporal verification of throughput constraints at the selected checkpoint.

5.1. Strategy overview

The overview of our novel throughput consistency verification strategy is depicted in Table I as follows.

At any candidate throughput checkpoint as defined in Definition 3, the first step is to determine whether the candidate checkpoint $S_p$ should be selected as a checkpoint or not by our throughput-based checkpoint selection strategy. If the result is true then throughput consistency verification is required. Otherwise, the strategy will move on until the system time arrives at the next candidate checkpoint $S_{p+1}$. For throughput consistency verification, it is to determine whether the current throughput consistency state $\alpha\%$ defined in Definition 6 is no less than the target throughput consistency state $\beta\%$ or not.

5.2. Throughput-based checkpoint selection

Definition 3 has defined the candidate throughput checkpoints along the system timeline. For example, given the upper bound temporal constraint $U(WF_i)$, if the basic observation time unit $bt$ is defined as 10\% of $U(WF_i)$, then there will be a set of ten candidate checkpoints along the system timeline. When the system time arrives at a candidate checkpoint, a checkpoint selection strategy needs to determine whether the current candidate should be selected as a checkpoint or not. As introduced in Section 2, there are many conventional response-time based checkpoint selection strategies, but none of them can be employed directly for throughput consistency monitoring. So far, there is only one intuitive equal distribution strategy $CSS_{TP}$ proposed in [5] where every candidate is selected as a checkpoint. $CSS_{TP}$ is a generic but preliminary solution. It tends to select more checkpoints than actually required. Meanwhile, in the original work of $CSS_{TP}$, it employs conventional random distribution models rather than the queuing model. Therefore, both its efficiency and effectiveness can be improved.

To further illustrate and evaluate the idea of throughput based checkpoint selection and improve the performance of $CSS_{TP}$ to address Challenge #3 mentioned in the introduction, we propose a novel QoS-aware throughput based checkpoint selection strategy (denoted as $CSS_{QoS}$).

Quality of service aware throughput based checkpoint selection strategy

Given the same batch of workflows in Definition 1, at a candidate throughput checkpoint $S_p$, the rule for the QoS-aware throughput based checkpoint selection strategy is defined as follows: if $TH_{S_{p-1}} \geq TH_{Cons}$, $S_p$ is not selected as a checkpoint. Otherwise, $S_p$ is selected as a checkpoint.

<table>
<thead>
<tr>
<th>Input</th>
<th>The workflow running state at a candidate checkpoint $S_p$; The target throughput consistency $\beta%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>The workflow throughput consistency state at $S_p$</td>
</tr>
<tr>
<td>Step 1</td>
<td>Throughput-based checkpoint selection</td>
</tr>
<tr>
<td></td>
<td>1) Given our throughput-based checkpoint selection strategy, determine whether $S_p$ should be selected as a throughput checkpoint or not;</td>
</tr>
<tr>
<td></td>
<td>2) If 1) is true, then continue to step 2 Else, break until the next candidate checkpoint $S_{p+1}$</td>
</tr>
<tr>
<td>Step 2</td>
<td>Throughput consistency verification</td>
</tr>
<tr>
<td></td>
<td>1) Given the runtime throughput consistency model, determine the current temporal consistency state $\alpha%$;</td>
</tr>
<tr>
<td></td>
<td>2) If $\alpha% \geq \beta%$ then break; Else, report a detected potential temporal violation.</td>
</tr>
</tbody>
</table>

Table I. A novel throughput consistency verification strategy.
Here, $TH\big|_{S_p/S_{p-1}}$ is calculated according to Definition 1. $TH_{Cons}\big|_{S_p/S_{p-1}}$ is the expected percentage of completion for a basic observation time unit $bt$ between $S_p$ and $S_{p-1}$ according to the deadline assignment strategy presented in Definition 2. For example, if $bt$ is specified as $v\%$ (e.g., 10\%) of $U(WF_i)$, then $TH_{Cons}\big|_{S_p/S_{p-1}}$ is equal to $v\%$ (e.g., 10\%).

6. EVALUATION

In this section, we evaluate our strategy CSSQoS and compare with five representative response-time based checkpoint selection strategies and one generic throughput-based checkpoint selection strategy. The basic idea of each strategy is described as follows:

- **CSS\text{all}**: It takes every activity as a checkpoint [22].
- **CSS\text{decision}**: It takes the start activity as a checkpoint and adds a checkpoint after each decision activity is executed [23].
- **CSS\text{max}**: It takes $a_i$ as a checkpoint if $R(a_i) > D(a_i)$ [24].
- **CSS\text{mean}**: It takes $a_i$ as a checkpoint if $R(a_i) > M(a_i)$ [25].
- **CSS\text{TD}**: It is a representative state-of-the-art response-time based checkpoint selection strategy, which can achieve necessity and sufficiency [15].
- **CSS\text{TP}**: It is a generic throughput based checkpoint selection strategy which means to take every candidate time point as a checkpoint given in Definition 3. Its preliminary version was presented in [5] mainly for the purpose of proof-of-concept, but it has been significantly concretized in this paper to take advantage of the new definitions and models.

In this paper, we do not intend to implement and compare all existing strategies because it is not possible or even necessary. A more complete list of response-time based checkpoint selection strategies can be found in [13]. Here, we try to select one or two representatives from each category of strategies based on their main ideas. Specifically, CSS\text{all} is an intuitive response-time based strategy, CSS\text{decision} is mainly based on the process structures, CSS\text{max} and CSS\text{mean} are based on simple statistics of activity durations, and CSS\text{TD} is a representative state-of-the-art response-time based strategy. Although there are several versions of the state-of-the-art strategies based on different temporal consistency models, their performance is similar because they can all select the same number of checkpoints as the number of local violations, namely, necessary and sufficient. Therefore, we only compare with CSS\text{TD} in this paper. As for CSS\text{TP}, it is the only existing throughput-based strategy available in the literature.

6.1. Experimental settings

Our experiments are conducted in SwinFlow-Cloud, a prototype cloud workflow system [36]. In our experiments, we simulate the continuous running of many batches of parallel workflows in the cloud. To simplify our experiments, all workflow instances are composed of only sequential activities. However, as mentioned in Section 3, our strategy can be easily applied to other process structures such as parallelism and choice with activity weight [1].

The basic experimental settings are described in Table II. As shown in Table II, we have designed three rounds of experiments each with five batches of workflows. The workflow size with 10 to 30 activities is very typical for business processes [18]. Therefore, workflows with an average of ten activities are tested in the first round, while larger ones with 20 and 30 activities are tested in the second and third round to evaluate the effect of workflow size on the performance of these strategies. The number of parallel processes increases from 100, 500, 1200, 5000, to 10,000 that covers a typical range of concurrent user requests in a real-time business system. In addition, three types of services, viz. small, medium, and large, are employed to accommodate different number of requests. Process structures including sequential and parallel processes are manually generated for

---

Please visit [http://swinflowcloud.thss.tsinghua.edu.cn/](http://swinflowcloud.thss.tsinghua.edu.cn/) for more details.
the processes in each batch. As for the activity durations, the mean execution time is first randomly
generated from (30, 300) time units to cover a large searching space using normal distribution
models. Afterwards, the queuing model proposed in [34] is adopted to simulate the actual system
performance. The basic observation time unit $bt$ is set as 5% or 10% of the expected work
flow completion time, that is, to generate 20 or 10 time points along the system timeline as candidate
checkpoints for throughput-based strategies. For response-time based strategies, their candidate
checkpoints are every activity points. Finally, a violation handling strategy is required to recover the
detected violations. There are many violation handling strategies with different capabilities and costs
[14]. In this paper, to focus on the evaluation of the temporal consistency monitoring strategies,
same as in our previous work [31], we adopt a simple pseudo violation handling strategy with 80%
success rate which can compensate for running delays in all situations. Concrete exception handling
strategies can be referred to [14, 28, 30]. All other unique settings for each checkpoint selection
strategy are the default values used in each reference and, hence, are omitted here. Each batch of
processes is running repeatedly for 1000 times to get the average values.

6.2. Experimental results

Here, we demonstrate the experimental results for the evaluation of both efficiency and effectiveness.
Full details of the experiments and other related materials can be found online.\(^1\)

6.2.1. Efficiency. In general, the efficiency of temporal consistency monitoring can be measured by
the time overhead of each strategy including both the computation and communication overhead. In

\(^1\)http://www.ict.swin.edu.au/personal/dyuan/doc/ICSSP14.rar
addition, because the number of selected checkpoints determines the number of possible violation handling, which dominates the cost of the entire temporal QoS framework, as in many studies [12, 13, 26], the efficiency of temporal consistency monitoring should also consider the number of selected checkpoints.

According to our experiments, the computation overhead of our strategy is very small (in milliseconds) and linear to the number of workflow activities. Considering most workflow activities are running in seconds or minutes, the computation overhead is trivial and, thus, can be neglected. This is consistent to the results of the previous work [15]. In contrast, the major time overhead is produced by the communication for acquiring the runtime information such as reading the workflow system log to obtain the start and end time of a workflow activity. Specifically, for conventional response-time based strategies, because every workflow activity is a candidate checkpoint, communication is required once at every activity point. In contrast, for throughput-based strategies, communication is only required once at each candidate system time point. Therefore, the number of communication required for response-time based strategies are much larger than that of throughput-based strategies. However, for response-time based strategies, the communication only requires to read the start and end times of one workflow activity, while the communication for throughput-based strategies needs to read the data of all workflow activities within the basic observation time unit. For example, the DATETIME type usually needs 8 bytes storage [37], and thus, to acquire the start time and end time of an activity, 16 bytes of data needs to be read. If there are 100 parallel processes each with ten workflow activities, and there are ten system time points, the question becomes ‘Which one is larger: the time overhead for 1000 communications each reads 16 bytes, or the time overhead for 10 communications each reads 1600 bytes?’ (Figure 1)

It is not easy to answer this question directly because the communication channel and network performance in each system environment can be very different. Therefore, we employ an analogy experiment using Amazon Web Service, which is one of the most popular public cloud computing platforms. A small EC2 instance (http://aws.amazon.com/ec2/) acting as the monitoring component is created to read a text file representing the system log stored in a S3 service (http://aws.amazon.com/s3/). We have recorded the communication time for reading different bytes of data from the same S3 file. Specifically, we generate two groups of test cases, the first group with an increment of 10 bytes from 10 bytes to 2 KB, and the second group with an increment of 10 KB from 10 KB to 2 MB. Each test case was repeated for ten times to get the average communication time. Because of the space limit, we only demonstrate a part of the testing results in Figure 2.

As shown in Figure 2, despite the huge difference in the data size between the two groups, their average reading time are 57.9 and 55.0 ms, respectively, namely very close to each other. Clearly, the reading time does not follow a linear relationship with the data size but appears to be very random with a large deviation. Because the communication process in the cloud is transparent to us, the reason we speculated is that the major overhead does not lie in the reading of data but other factors such as reading the metadata to locate the data file, selecting one of the replicas and creating the communication channels. As for large deviations, they are probably caused by the randomness in network performance. Specifically, the reading time for 16 bytes is 73.1 ms, and the reading time

![Figure 1. Runtime throughput consistency model.](image-url)
for 1600 bytes is 76.1 ms in our experiments. Meanwhile, because response-time based strategies need 1000 communications, while throughput-based strategies only need ten communications, this is nearly 100 times reduction in the communication overhead. Therefore, based on such a result, we can conclude that the communication overhead for a throughput-based strategy is much smaller than that of a response-time based strategy. Next, we compare the number of selected checkpoints, which is another important measurement for efficiency.

Figure 3 depicts the average number of checkpoints selected by each strategy in Round 1. It is not surprising to see that there is a big gap between the response-time-based strategies, and the throughput-based strategies as the former is working on activity points while the latter is working on system time points.

For example, in batch 1 with 100 parallel processes and each with an average of ten activities, CSS\textsubscript{all} has selected 1000 checkpoints because it takes every activity as a checkpoint. CSS\textsubscript{decision} selects the start, the second, the fifth, and the seventh activity because they are either the fork or joint activities for decision making. CSS\textsubscript{TD} only selects necessary and sufficient checkpoints, and thus, the number of checkpoints for CSS\textsubscript{TD} is equal to the number of local temporal violations. As for other response-time based strategies, some quantitative selection criteria are employed so that a large amount of checkpoints can be reduced. Compared with CSS\textsubscript{all}, the average reduction rate for CSS\textsubscript{decision}, CSS\textsubscript{max}, CSS\textsubscript{mean}, and CSS\textsubscript{TD} are 60%, 83%, 55%, and 68%, respectively. As for throughput-based checkpoint selection strategies, because they are working on system time points, there are only ten candidate checkpoints along the system timeline when $bt$ is set as 10% of the total workflow.
duration. Therefore, $CSS_{TP}$ selects ten checkpoints as it selects all candidates, while $CSS_{QoS}$ only selects half of the candidates. Compared with $CSS_{all}$, the average reduction rates for $CSS_{TP}$ and our QoS-aware strategy $CSS_{QoS}$ are 99.7% and 99.8%, respectively, which is remarkable. The results for other two rounds of experiments are similar. Because of the page limit, more details are omitted here but can be found in our online documents.

6.2.2. Effectiveness. Effectiveness is measured by how close the real on-time completion rate (denoted as $\alpha$ %) from the target on-time completion rate (denoted as $\beta$ %, namely, the service quality). As mentioned in the introduction, in a cloud computing environment, higher service quality does not necessarily mean a better effectiveness because the service provider needs to cover the cost of over-provisioned resources. Therefore, the best strategy is the one with the on-time completion rate most close to the target. To measure and compare closeness, the value of closeness is defined as follows.

Closeness:

$$1 - |\alpha - \beta|\%$$

Figure 4 shows the measurement of closeness for each strategy in round 1. Because all of the strategies achieve the on-time completion rate higher than the target 90% but none is achieving exactly the target, the service quality has been over provisioned.

Specifically, based on Formula 4, $CSS_{decision}$ has the highest closeness with an average of 96%, $CSS_{TP}$ and $CSS_{QoS}$ have the similar closeness with an average of 95.1% and 94.8%, respectively, while the rest of the strategies have similar closeness with an average around 90%. For response-time based strategies, as shown in Figure 2, most response-time based strategies except $CSS_{max}$ select more checkpoints than $CSS_{TD}$, namely, more violation handling would have been conducted than actually required. Therefore, because the on-time completion rate for $CSS_{TD}$ is close to 100%, the on-time completion rates for other response-time based strategies are also close to 100%, namely, a closeness value of 90%. However, $CSS_{decision}$ is an exception as it only works on fixed activity points. Therefore, although it selects more checkpoints than $CSS_{TD}$, many temporal violations are missed. In contrast, $CSS_{max}$ selects much fewer checkpoints than $CSS_{TD}$, but can still achieve similar on-time completion rate. Such a result shows that not every local temporal violation needs to be handled, which is consistent with the results presented in [31]. A violation handling point selection strategy can be employed to address this problem. However, because this paper focuses on checkpoint selection, every local temporal violation is handled by our pseudo violation handling strategy.

The results for round 2 are depicted in Figure 5. For round 2, workflows with an average of 20 activities are tested. The results show that $CSS_{TP}$ and $CSS_{QoS}$ have the similar highest closeness of 95.5% and 95.6%, respectively, while the average is still around 95%. The results for response-time based strategies are also very similar as in round 1 with an average of 90% except that the closeness
of $CSS_{\text{decision}}$ dropped from 96% to 91% as its on-time completion rate increased from 94% to 99%. This is not surprising because more decision points are distributed for large size workflows, and hence, there are more chances to cover the necessary and sufficient checkpoints.

Figure 6 shows the results for round 3 where workflows with an average of 30 activities are tested. With an average closeness of 90%, the values of closeness for all response-time based strategies have leveled off in this round. In comparison, $CSS_{\text{QoS}}$ has the best closeness of 96.6%, which is better than $CSS_{\text{TP}}$ with its best closeness of 94.9%. However, their average closeness are 94.7% and 95.1%, respectively, similar as in the last two rounds.

The above results are tested with a basic observation time unit $bt$ as 10% of the expected workflow completion time, that is, with 10 candidate time points. In addition, we also test and compare the performance $CSS_{\text{TP}}$ and $CSS_{\text{QoS}}$ with a smaller $bt$, which is equal to 5% of the expected workflow completion time, that is with 20 candidate time points. Specifically, with the same settings as in round 1, the results show that the number of checkpoints has been doubled for both $CSS_{\text{TP}}$ and $CSS_{\text{QoS}}$. However, the increase of the on-time completion rate is less than 0.5%. Such a result actually shows that more checkpoints do not guarantee a higher on-time completion rate because only necessary and sufficient checkpoints are required. Accordingly, there is a minimum for the number of candidate time points to ensure the target service quality. However, it is very difficult, if not impossible, for us to determine such a value because our strategy cannot achieve necessity and sufficiency at this stage. This is actually one of the most important directions for our future work.

To summarize, the results of the three rounds of experiments are similar in general, which show that the workflow size has limited impact on the performance of checkpoint selection. This is consistent with the conclusion found in [31]. In general, throughput-based checkpoint selection strategies including both $CSS_{\text{TP}}$ and $CSS_{\text{QoS}}$ show a remarkable reduction in the number of checkpoints (namely, higher efficiency) and a better closeness (namely, higher effectiveness) than all response-time based strategies. Specifically, $CSS_{\text{QoS}}$ achieved the best average checkpoint reduction rate of
99.8% and the best average closeness of 95.0% in the three rounds of experiments. Therefore, throughput-based strategies have significant advantages over conventional response-time based strategies in monitoring large batch of parallel processes, and our strategy CSS_QoS achieves the best efficiency and effectiveness among all of them. Furthermore, it is evident that currently no strategies can reach nearly 100% closeness because they all achieved an on-time completion rate over the targeted 90%, which means the service quality has been over provisioned. Therefore, there is still room to improve in the future.

6.3. Threats to validity

Here we briefly discuss the threats to validity in our work. We discuss the external threats followed by the internal threats.

6.3.1. External threats to validity. The main threat to the external validity of our work is the representativeness of the three big challenges that represent the major differences between scientific workflows running in a community-based and business processes running in a market-oriented computing environment. However, there are many other differences such as more and less with human intervention as discussed in the literatures [16, 38]. We focus on these three differences as they have major impacts on workflow temporal verification. In our previous work [39], a pulsar-searching scientific workflow and a security-exchange business workflow are discussed as motivating examples, which support the representativeness of the three major differences. In addition, this paper focuses on business processes with computational tasks where the underlying processing units are virtual machines, but real-world business processes can be much more diverse and complicated where both computational and human resources are included. Therefore, our strategy cannot be directly applied to generic business processes including the software process before we change the underlying resource model and adapt our strategy. In the future, we can further minimize the external threats by investigating more real-world examples so as to validate the challenges and extend the resource models to facilitate the design of workflow temporal verification strategies for more generic business processes.

6.3.2. Internal threats to validity. The main threat to the internal validity of our evaluation is the comprehensiveness of our experiments. As shown in Table II, many parameters and models have been used to generate the testing cases and evaluate the performance of our strategy. To guarantee the representativeness of the experimental settings and also for the comparison purpose, the setting of the parameters is based on either similar settings or statistics obtained in the earlier work such as [5, 31, 33] or real-world scenarios such as security exchange business processes and Amazon Web Services public cloud. Meanwhile, different rounds of experiments with a variety of settings on such as activity durations, number of parallel processes and number of cloud services have been conducted. Therefore, our experiments have explored a representative and large enough searching space to validate that throughput-based strategies are generally better than response-time based strategies in monitoring a large batch of parallel processes. However, as the research on the temporal verification for business processes is still at its early stage, some settings such as deadline assignment and violation handling are not comprehensive enough as they are still open questions. In addition, our simulation experiments are conducted in the cloud-based prototype workflow system SwinFlow-Cloud [36] which is mainly for research purpose at this stage. Therefore, with its continuous improvement and future deployment in real-world business organizations, we can further minimize the internal threats to validity by testing the proposed strategies directly with real-world processes.

7. CONCLUSION AND FUTURE WORK

Temporal verification plays an important role in achieving on-time completion of scientific and business workflows. However, because of three big challenges including #1: single process versus a large batch of parallel processes, #2: dedicated and static versus shared and dynamic resource
environments, and #3: best-effort versus strict service quality constraints, current response-time based checkpoint selection and temporal verification strategies for a single scientific workflow cannot be applied directly to the monitoring of a large batch of parallel business processes. To address these challenges, we have investigated the measurement of workflow throughput instead of conventional response time for the monitoring of business processes. Specifically, challenge #1 is addressed by our new workflow throughput model, challenge #2 is addressed by the queueing model, and challenge #3 is addressed by our proposed novel QoS-aware throughput based checkpoint selection strategy. Comprehensive experimental results have demonstrated that throughput-based strategies are generally better than response-time based strategies in monitoring a large batch of parallel processes. Specifically, our strategy can achieve the best efficiency (namely, the best average reduction in the number of checkpoints) and the best effectiveness (namely, the best average closeness to the target service quality).

In the future, we will further investigate and improve throughput-based temporal verification for a large batch of parallel business processes. For example, the checkpoint selection strategy can be further improved by incorporating more runtime information. Meanwhile, fine-grained violation handling strategies can be designed so as to recover the detected violations but not over-provision the service quality. The ultimate goal is to achieve a ‘necessary and sufficient’ throughput-based checkpoint selection strategy, which can achieve the target service quality without either under or over service provisioning. Finally, we will try to explore the scenarios where both computational and human resources are included in the business processes so as to extend our studies to more generic business processes including the software process.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable comments to improve this paper. The research work reported in this paper is partly supported by National Natural Science Foundation of China (NSFC) under no. 61300042 and no. 61021004, Australian Research Council under LP0990393 and LP130100324, the Fundamental Research Funds for the Central Universities, and Shanghai Knowledge Service Platform Project under no. ZF1213.

REFERENCES


31. Liu X, Yang Y, Yuan D, Chen J. Do we need to handle every temporal violation in scientific workflow systems? *ACM Trans. on Software Engineering and Methodology* 2014; 23(1), Article 5, Feb.


