Workflow Temporal Verification for Monitoring Parallel Business Processes*

Xiao Liu$^{1,2}$, Dingxian Wang$^1$, Dong Yuan$^3$, Futian Wang$^4$, Yun Yang$^5$

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

xliu@sei.ecnu.edu.cn, dingxianwang@gmail.com, {dyuan,yang}@swin.edu.au, wft@ahu.edu.cn

ABSTRACT
Workflow 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 due to significant time overhead. To address such a problem, based on a novel runtime throughput consistency model, this paper proposes a QoS-aware throughputs 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 well as other representative response-time based checkpoint selection strategies.

Keywords
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 [20, 21]. 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, e.g. the cloud workflow system [21]. In the meantime, cloud computing is establishing itself as a new paradigm for delivering IT infrastructure elements such as computing, storage and network resources as IT services over the Internet [16]. 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 [2, 23, 34, 36]. However, due to its dynamic nature, to guarantee the delivery of satisfactory service quality is a big challenge. There are many efforts from both Software Engineering [11, 33] and Parallel and Distributed Computing areas [15] dedicated to the quality assurance of cloud and general web services [31].

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 constraint setting, temporal-aware service selection, temporal consistency monitoring and temporal violation handling [22]. 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 [8]. 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 [27].

In this paper, we focus on temporal consistency monitoring for a large batch of parallel business processes running in a cloud computing environment. 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 [28]. However, to deal with business processes, current strategies for scientific workflows may become much less useful due to their significant differences [3]. Specifically, there are three big challenges as follows.

Challenge #1: Single process vs. 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 workflows 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 workflows is how to efficiently measure the temporal consistency states of a large batch of parallel processes.

Challenge #2: Dedicated and static vs. 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 [20]. In contrast, with the emergence of cloud computing, business workflows 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 vs. 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 workflows running in the cloud have strict constraints on time, cost, reliability and other QoS requirements since 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 [16]. 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 workflows but without over provisioning cloud resources [19].

Given the above three big challenges, conventional temporal consistency monitoring strategies for single large-scale scientific workflows cannot be applied directly to business workflows. 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 which 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 as well as 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 [1, 16, 21, 22, 26, 32, 38]. In recent years, with the emergence 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 [15, 33]. 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 [22] 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 [5] takes every workflow activity as a checkpoint. The work in [30] 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 which selects an activity as a checkpoint if its execution time exceeds the maximum duration and the one which selects an activity as a checkpoint if its execution time exceeds the mean duration [9, 10]. The checkpoint selection which satisfies the property of necessity and sufficiency is proposed in [6] 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 [8] 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 [7, 13, 17, 37]. Recently, a continuous-state based temporal consistency model is proposed which can measure the temporal consistency state using confidence values such as 90% [20]. 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 [24].

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 so far very limited work investigates the monitoring of a large batch of parallel business workflows. Our latest work in [23] 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 [34]. Workflow throughput, namely the throughput of a workflow system, is the number of workflows that have been completed in a basic observation time unit [1]. Workflow throughput is a better measurement to address Challenge #1 as mentioned in the introduction. 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 [18]. For example, if one workflow activity running for 2 minutes and another one running for 20 seconds are both completed during the same observation time unit, their contributions to the system throughput are treated the same, i.e. 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 [23].

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 [7]. 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 [6], 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 [25].

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 [20]; $WF_i$ 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 $Batch(WF_1,WF_2,\ldots,WF_q)$ 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 $a\{\}^{S_{t-1}}_{S_t}$, then the current system throughput is defined as $TH(S_t) = W \times M(a\{\}^{S_{t-1}}_{S_t})/T$ 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 0 which means the business process starts at system time 0. 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 [19].

Definition 2 (An Example Workflow Throughput 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 $S_i$, where $S_i - S_0 = n \times bt$ ($n = 1,2,3,\ldots, U(WF_i)/bt - 1$), the throughput constraint assigned at $S_i$ by a throughput constraint setting strategy is $Cons(S_i) = THCons(S_i) = W \times M(a\{\}^{S_{i-1}}_{S_i})/T$ which means that the expected accumulated system throughput $\sum_{i=1}^{T} TH(S_i)$ 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, since there is normally a basic time unit for system monitoring, i.e. $bt$, 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 $S_i$, along the workflow execution timeline is a candidate throughput checkpoint if $S_i - S_0 = k \times bt$ ($k = 1,2,3,\ldots, U(WF_i)/bt - 1$).

As preliminarily discussed in [23], 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 [20]. 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 \( E(a_i) \), i.e. \( \mu_i \), is estimated using a queuing model which is a latest work on the performance analysis of cloud computing services [14]. 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 FCFS. The capacity of the system is \( m+r \) which means that the buffer size for incoming request is equal to \( r \), i.e. \( n-m \) 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 [14] 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 [12]. 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:

\[
\begin{align*}
\text{Max} \left( TH \left[ \frac{U(WF_i)}{S_p} \right] \right) &= \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \times \frac{q \times U(WF_i)}{W \times d(a[1] \mid L(S_p))} \quad (1) \\
\text{Exp} \left( TH \left[ \frac{U(WF_i)}{S_p} \right] \right) &= \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \times \frac{q \times U(WF_i)}{W \times E(a[1] \mid L(S_p))} \quad (2) \\
\text{Min} \left( TH \left[ \frac{U(WF_i)}{S_p} \right] \right) &= \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \times \frac{q \times U(WF_i)}{W \times D(a[1] \mid L(S_p))} \quad (3)
\end{align*}
\]

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
\[
\text{TH} \left[ \frac{S_p}{S_0} \right] + \text{Max} \left( \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \right) \geq 100\% ;
\]

2) **Absolute Inconsistency (AI)**, if
\[
\text{TH} \left[ \frac{S_p}{S_0} \right] + \text{Max} \left( \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \right) \leq 100\% ;
\]

3) **\( \alpha\% \) Consistency (\( \alpha\% C \)), if
\[
F(\lambda_{\alpha}) = \text{TH} \left[ \frac{S_p}{S_0} \right] + \text{Exp} \left( \text{TH} \left[ \frac{U(WF_i)}{S_p} \right] \right)
\]

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 \( F(\lambda_{\alpha}) = \frac{1}{\sqrt{2\pi}} \int_{-\lambda_{\alpha}}^{\lambda_{\alpha}} \frac{-e^{-t^2/2}}{2\pi} dt = \alpha\% \).

![Fig. 1. Runtime Throughput Consistency Model](attachment:image)
According to the “3σ 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 [24, 28]. 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 is below the target. Please refer to [27] 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 which 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 1 as follows.

<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>

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 \(b_t\) is defined as 10% of \(U(WF_i)\), then there will be a set of total 10 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 [23] 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 \(CSS_{QoS}\)).

**QoS-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 \(TH \left[ S_p \right]_{S_{p-1}} \geq THCons \left[ S_p \right]_{S_{p-1}}, S_p\) is not selected as a checkpoint. Otherwise, \(S_p\) is selected as a checkpoint.

Here, \(TH \left[ S_p \right]_{S_{p-1}}\) is calculated according to Definition 1. \(THCons \left[ S_p \right]_{S_{p-1}}\) is the expected percentage of completion for a basic observation time unit \(b_t\) between \(S_{p-1}\) and \(S_p\) according to the deadline assignment strategy presented in Definition 2. For example, if \(b_t\) is specified as \(\nu%\) (e.g. 10%) of \(U(WF_i)\), then \(THCons \left[ S_p \right]_{S_{p-1}}\) is equal to \(\nu%\) (e.g. 10%).
6. EVALUATION
In this section, we evaluate our strategies \( CSS_{QoS} \) and compare with 5 representative response-time based checkpoint selection strategies and 1 generic throughput based checkpoint selection strategy. The basic idea of each strategy is described as follows:

- **CSS\(_{all}\)**: it takes every activity as a checkpoint [5].
- **CSS\(_{decision}\)**: it takes the start activity as a checkpoint and adds a checkpoint after each decision activity is executed [30].
- **CSS\(_{max}\)**: it takes \( a_i \) as a checkpoint if \( R(a_i) > D(a_i) \) [9].
- **CSS\(_{mean}\)**: it takes \( a_i \) as a checkpoint if \( R(a_i) > M(a_i) \) [10].
- **CSS\(_{TD}\)**: it is a representative state-of-the-art response-time based checkpoint selection strategy which can achieve necessity and sufficiency [28].
- **CSS\(_{TP}\)**: it is a generic throughput based checkpoint selection strategy which means to take every candidate time point as a checkpoint given Definition 3. Its preliminary version was presented in [23] 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 [8]. Here, we try to select one or two representatives from each category of strategies based on their main ideas. Specifically, **CSS\(_{all}\)** is an intuitive response-time based strategy; **CSS\(_{decision}\)** is mainly based on the process structures; **CSS\(_{max}\)** and **CSS\(_{mean}\)** are based on simple statistics of activity durations; **CSS\(_{TD}\)** 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 since 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\(_{TD}\)** in this paper. As for **CSS\(_{TP}\)**, it is the only existing throughput based strategy available in the literature.

6.1 Experimental Settings
Our experiments are conducted in SwinFlow-Cloud\(^1\), a prototype cloud workflow system [4]. 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 [20].

The basic experimental settings are described in Table 2. As shown in Table 2, we have designed 3 rounds of experiments each with 5 batches of workloads. The workflow size with 10 to 30 activities is very typical for business processes [1]. Therefore, workflows with an average of 10 activities are tested in the first round while larger ones with 20 and 30 activities are tested in the second and the 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 10000 which 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 are manually generated for 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 [14] is adopted to simulate the actual system performance. The basic observation time unit \( bt \) is set as 5% or 10% of the expected workflow completion time, i.e. 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 [27]. In this paper, to focus on the evaluation of the temporal consistency monitoring strategies, same as in our previous work [24], 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 [13, 27, 37]. All other unique settings for each checkpoint selection strategy are the default values used in each reference and hence omitted here. Each batch of processes is running repeatedly for 1,000 times to get the average values.

### Table 2. Experimental Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Round 1</strong></td>
<td></td>
</tr>
<tr>
<td>Resource Settings</td>
<td></td>
</tr>
<tr>
<td>Number of Parallel Processes</td>
<td>100, 500, 1200, 5000, 10000</td>
</tr>
<tr>
<td>Number of Cloud Services</td>
<td>20, 100, 200, 800, 1500</td>
</tr>
<tr>
<td><strong>Round 2</strong></td>
<td></td>
</tr>
<tr>
<td>Resource Settings</td>
<td></td>
</tr>
<tr>
<td>Number of Parallel Processes</td>
<td>100, 500, 1200, 5000, 10000</td>
</tr>
<tr>
<td>Number of Cloud Services</td>
<td>15, 60, 130, 550, 1000</td>
</tr>
<tr>
<td><strong>Round 3</strong></td>
<td></td>
</tr>
<tr>
<td>Resource Settings</td>
<td></td>
</tr>
<tr>
<td>Number of Parallel Processes</td>
<td>100, 500, 1200, 5000, 10000</td>
</tr>
<tr>
<td>Number of Cloud Services</td>
<td>10, 50, 110, 420, 660</td>
</tr>
<tr>
<td><strong>Service Provision</strong></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>Small</td>
</tr>
<tr>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td><strong>Workload Settings</strong></td>
<td></td>
</tr>
<tr>
<td>Process Structure</td>
<td>5 Batches of processes are generated in each round of experiment. The process structure in the same batch is identical which is manually generated.</td>
</tr>
<tr>
<td>Activity Durations</td>
<td>Step 1: generate the basic statistics for activity execution time using normal distribution models. Step 2: generate the activity durations using the queuing model as described in Definition 4.</td>
</tr>
<tr>
<td>Deadline Assignment</td>
<td>The deadline assignment strategy proposed in [18] is adopted in this paper where a confidence value of 90% is specified, namely the target on-time completion rate 90% is 90%.</td>
</tr>
<tr>
<td>Candidate Checkpoints</td>
<td>1) For throughput based strategies, the candidate checkpoints are time points as defined in Definition 3, and the basic observation time unit ( bt ) is specified as 5% or 10% of the expected workflow completion time. 2) For response-time based strategies, the candidate checkpoints are activity points.</td>
</tr>
<tr>
<td>Violation Handling</td>
<td>Similar as in our previous work [23], a pseudo violation handling strategy with 80% success rate is adopted which can recover detected violations by compensating for all running delays.</td>
</tr>
</tbody>
</table>

---

\(^1\) Please visit http://www.ict.swin.edu.au/personal/dcao/ for more details.
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 online³.

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 addition, since 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 [6, 8, 22], 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 [28]. 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, since 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 [39], 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 10 workflow activities, and there are 10 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?”

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 2K, and the second group with an increment of 10K from 10K to 2M. Each test case was repeated for 10 times to get the average communication time. Due to 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 milliseconds and 55.0 milliseconds 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. Since 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 milliseconds and the reading time for 1600 bytes is 76.1 milliseconds in our experiments. Meanwhile, since response-time based strategies need 1000 communications while throughput based strategies only need 10 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 10 activities, CSS_{all} has selected 1000 checkpoints since it takes every activity as a checkpoint. CSS_{decision} selects the start activity, and the second, the fifth and the seventh activity since they are either the fork or joint activities for decision making. CSS_{TP} only selects necessary and sufficient checkpoints and thus

the number of checkpoints for $CSS_{TD}$ 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 $CSS_{all}$, the average reduction rate for $CSS_{decision}$, $CSS_{max}$, $CSS_{mean}$, and $CSS_{TD}$ are 60%, 83%, 55%, 68% respectively. As for throughput based checkpoint selection strategies, since they are working on system time points, there are only 10 candidate checkpoints along the system timeline when $bt$ is set as 10% of the total workflow duration. Therefore, $CSS_{TP}$ selects 10 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. Due to 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.

$$Closeness: 1 - |\alpha - \beta|\%$$  \quad (4)

Figure 4 shows the measurement of closeness for each strategy in Round 1. Since 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.

Fig. 4. Measurement of Closeness for each Strategy (Round 1)

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, since 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 [24]. A violation handling point selection strategy can be employed to address this problem. However, since this paper focuses on checkpoint selection, every local temporal violation is handled by our pseudo violation handling strategy.

Fig. 5. Measurement of Closeness for each Strategy (Round 2)

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 $CSS_{decision}$ 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.

Fig. 6. Measurement of Closeness for each Strategy (Round 3)

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, $CSS_{QoS}$ has the best closeness of 96.6% which is better than $CSS_{TP}$ 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, i.e. with 10 candidate time points. In addition, we also test and compare the
performance $CSS_{TP}$ and $CSS_{QoS}$ with a smaller $bt$ which is equal to 5% of the expected workflow completion time, i.e. 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 $CSS_{TP}$ and $CSS_{QoS}$. 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 since 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 since 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 3 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 [24]. In general, throughput based checkpoint selection strategies including both $CSS_{TP}$ and $CSS_{QoS}$ 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, $CSS_{QoS}$ achieved the best average checkpoint reduction rate of 99.8% and the best average closeness of 95.0% in the 3 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 since 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.

External threats to validity. The main threat to the external validity of our work is the representativeness of the three big challenges which represent the major differences between scientific workflows and business workflows running in a community based and market oriented computing environments. However, there are many other differences such as more and less with human intervention as discussed in the literatures [3, 35]. We focus on these three differences as they have major impacts on workflow temporal verification. In our previous work [29], a pulsar searching scientific workflow and a security exchange business workflow are discussed as motivating examples, which supports the representativeness of the three major differences. In the future, we can further minimize the external threats by investigating more real-world examples and evaluating more affecting factors so as to validate or modify the challenges to guide the design of temporal verification strategies.

Internal threats to validity. The main threat to the internal validity of our evaluation is the comprehensiveness of our experiments. As shown in Table 2, 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[23, 24, 25] or real-world scenarios such as security exchange business workflows and AWS 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 workflows 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 the future, we can further minimize the internal threats to validity by incorporating other components and running the test cases in our cloud based workflow system SwinFlow-Cloud [4].

7. CONCLUSION AND FUTURE WORK

Temporal verification plays an important role in achieving on-time completion of scientific and business workflows. However, as discussed in the introduction section, due to the three big challenges, 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 such an issue, we have investigated the measurement of workflow throughput and presented a novel runtime throughput based temporal consistency model. Based on such a model, a novel throughput consistency verification strategy which includes a QoS-aware throughput based checkpoint selection strategy has been proposed. 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.

8. ACKNOWLEDGEMENTS

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 No. ZF1213.
9. REFERENCES


