Selecting Checkpoints along the Time Line: A Novel Temporal Checkpoint Selection Strategy for Monitoring a Batch of Parallel Business Processes

Xiao Liu, Yun Yang, Dahai Cao, Dong Yuan,
Swinburne University of Technology, Melbourne, Australia
{xliu, yyang, dcao, dyuan}@swin.edu.au

ABSTRACT
Nowadays, most business processes are running in a parallel, distributed and time-constrained manner. How to guarantee their on-time completion is a challenging issue. In the past few years, temporal checkpoint selection which selects a subset of workflow activities for verification of temporal consistency has been proved to be very successful in monitoring single, complex and large size scientific workflows. An intuitive approach is to apply those strategies to individual business processes. However, in such a case, the total number of checkpoints will be enormous, namely the cost for system monitoring and exception handling could be excessive. To address such an issue, we propose a brand new idea which selects time points along the workflow execution time line as checkpoints to monitor a batch of parallel business processes simultaneously instead of individually. Based on such an idea, a set of new definitions as well as a time-point based checkpoint selection strategy are presented in this paper. Preliminary results demonstrate that it can achieve an order of magnitude reduction in the number of checkpoints while maintaining satisfactory on-time completion rates compared with the state-of-the-art activity-point based checkpoint selection strategy.

Keywords
Checkpoint Selection, Temporal Verification, Parallel Processes

1. INTRODUCTION AND RELATED WORK
With the rapid growth of e-government and e-business, government agencies and enterprises are often required to process large numbers of customer requests in a constrained period of time. Typical examples include business processes for processing hundreds of thousands of tax declarations every day in a government taxation office during the peak period of tax declaration, and millions of transactions every minute in a stock exchange corporation during the peak time of the stock market [6]. Meanwhile, these business processes are often time constrained, for instance, the tax return process for each client request may need to be finished within 2 weeks, and the clearing process and money transfers in the stock market may need to be finished before 3:00am each weekday. Failures of on-time completion will result in significant deterioration of customer satisfaction and even huge financial losses. Therefore, timely completion of business processes is critical for delivering satisfactory services. Recently, 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 [4]. Customers such as government agencies and enterprises can access these services for running their software applications in a pay-as-you-go fashion while avoiding huge capital investment, energy consumption, and system maintenance of their own IT infrastructures. Cloud computing can offer powerful, on-demand and elastic computing resources which is an ideal hosting environment for running of a large batch of parallel business processes [9]. However, due to its dynamic nature, to guarantee the delivery of satisfactory service quality such as on-time completion is a big challenge. There are many efforts from both Software Engineering [8] and Distributed and Parallel Computing areas [4] dedicated to the quality assurance of cloud services.

In the last few years, temporal checkpoint selection which selects a subset of workflow activities for the verification of temporal consistency states, namely temporal verification, has been proved to be very successful in monitoring single, complex and large size scientific workflows. Temporal verification is generally rooted in the area of software verification and validation. Here, we briefly introduce some of the terms. “Temporal consistency” means that the runtime workflow execution state satisfies the temporal constraints assigned at workflow build time which can be either the final deadline or a milestone. “Temporal verification” is the process to determine the temporal consistency states by checking the workflow execution state at a selected activity point against a specific temporal constraint. For example, if the deadline of a workflow instance is set as 3:00PM at build time, and activity $a_p$ of the workflow instance which has been completed at 1:00PM is selected as a checkpoint at runtime, then if the rest of the workflow after $a_p$ can be finished within 2 hours, the workflow execution state is said to be of temporal consistency. Otherwise, it is said to be of temporal inconsistency, or namely, a temporal violation has occurred. In general, after a temporal violation is detected, exception handling strategies will be triggered by the workflow system to compensate for the time delays. Clearly, temporal checkpoint selection plays an important role in the above monitoring and control process because it determines where and how many times temporal verification and exception handling are required along the workflow execution. It has been proved by existing works that the larger the number of selected checkpoints, the higher the monitoring cost and exception handling cost [5]. Therefore, to select as few as possible number of checkpoints while maintaining satisfactory on-time completion rate is the major design objective for every checkpoint selection strategy.

In recent years, many checkpoint selection strategies, from intuitive rule based to sophisticated model based, have been proposed. For example, we can take every workflow activity as a checkpoint [3], or select the start activity as a checkpoint and add a new checkpoint after each decision activity is executed [7]. There are also 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE2013, May 18–26, 2013, San Francisco, CA, USA
Copyright 2013 ACM 978-1-59593-995-1...$5.00.
checkpoint if its execution time exceeds the mean duration. The checkpoint selection which satisfies the property of necessity and sufficiency named CSSPD is proposed in [1]. As proved, compared with all other existing strategies, it selects the minimum number of necessary and sufficient checkpoints and can guarantees the lowest violation rate by handling all detected temporal violations [5].

The state-of-the-art necessary and sufficient checkpoint selection strategies have been proved to be very successful in the monitoring of single, complex and large size scientific workflows. Intuitively, these strategies can be applied simultaneously to each individual process, and if every one of them is completed in time, then the entire batch of business processes are completed in time. However, the existing strategies cannot be applied directly to a large batch of business processes simply because the total number of checkpoints will be enormous. For example, if the batch contains one thousand parallel processes, then the total number of checkpoints will be around one thousand times the number of checkpoints for an individual process. As a result, the cost for system monitoring will also be increased drastically. In addition, when multiple checkpoints are selected at the same time (namely multiple temporal violations are detected), exception handling strategies such as the provision of additional resources will be requested for multiple times. However, since these batched processes are usually executed in a shared resource environment such as cloud computing, the provision of a single resource may well be enough to handle multiple temporal violations. Therefore, checkpoint selection for individual processes can also result in a huge waste of unnecessary exception handling cost.

Based on the above analysis, it is obvious that what we need for monitoring a large batch of parallel business processes is an efficient and cost-effective checkpoint selection strategy which can monitor all the processes in a simultaneous rather than individual fashion. However, as long as the target of checkpoint selection is the activity point of a single workflow instance, the problems mentioned above will be inevitable. To address such an issue, in this paper, we propose a novel idea which selects time points along the workflow system timeline as the checkpoints instead of workflow activities to monitor a batch of parallel business processes at the same time. For example, if the batch of business processes starts at 12:00PM and has a final deadline of 3:00PM, the selected checkpoints by our novel idea will be a set of time points such as {12:05PM, 12:10PM, 12:15PM,...,2:55PM}. At each time point, the overall temporal consistency state for the batch of processes can be evaluated with the novel throughput consistency model. Based on such an idea, a set of new definitions including the throughput, the candidate throughput checkpoint and the throughput consistency model will be first presented in this paper. Afterwards, we propose a time-point based checkpoint selection strategy which selects any time point as a checkpoint when the current system throughput is smaller than the expected mean throughput. Preliminary results demonstrate that it can achieve an order of magnitude reduction in the number of checkpoints while maintaining satisfactory on-time completion rates compared with other existing checkpoint selection strategies.

The reminder of this paper is organised as follows. Section 2 proposes the novel strategy. Section 3 presents the preliminary experimental results. Section 4 addresses the conclusions and points out the future work.

2. PROPOSED STRATEGY

The core idea of this work is to select time points instead of activity points as checkpoints for monitoring a batch of parallel processes. Those parallel processes may or may not start at the same system time point but have to be completed at the same time, i.e. the same deadline. Figure 1 illustrates a small segment of the trace for running a batch of parallel processes as an example scenario. Each dot in Figure 1 denotes a workflow activity where the red dots are checkpoints selected by the conventional activity-point based strategy. As can be seen in the figure, there are many selected checkpoints along the execution trace for the parallel processes. In contrast, our time-point based strategy only selects a few time points along the time line as checkpoints (denoted by red triangles and marked with $S_p$ to $S_{p+T}$) since it can monitor all the processes simultaneously. Each system time point (denoted as a small vertical bar along the time line and the distance between each time point is a basic time unit, e.g. one minute) is a candidate checkpoint and our novel strategy will decide which one should be selected. In Figure 1, for easy demonstration, we illustrate a simple case where the checkpoints are equally distributed but their actual distribution can be very dynamic.

![Figure 1. Activity-Point Based vs. Time-Point Based Strategy](image)

Our novel strategy does not require any additional information than the conventional strategy. However, a set of basic definitions for conventional activity-point based checkpoint selection need to be modified first. These basic definitions include the throughput, the throughput checkpoint and the throughput consistency model. Please be noted that since the primary aim of this paper is to present the new ideas and some initial results, we try to simplify the data, models and methods used in this paper as much as possible and leave the complex extensions as future work (detailed in Section 4). In the conventional work, the basic monitoring object is the response time of each workflow activity. However, for the monitoring of a large batch of parallel processes, the system throughput which measures how many workflow activities are completed per time unit is a better monitoring object because it can reflect the execution states of the entire batch of processes. Furthermore, the conventional definition of system throughput does not distinguish workflow activities with different completion time. For example, it is evident that the completion of a workflow activity running for 2 hours and the completion of another activity running for 10 minutes have significant differences in their contribution to meeting the final deadline. Therefore, we modify the definition for the system throughput to explicitly represent such a difference. Here are some basic annotations: $a_i$ is a workflow activity with its mean (i.e. expected) and runtime (i.e. real) durations denoted as $M(a_i)$ and $R(a_i)$ respectively; $P_j$ is a business process with its mean and runtime completion time denoted as $M(P_j)$ and $R(P_j)$ respectively; the basic time unit for monitoring is denoted as $bt$.
Definition 1 (Throughput). Given a batch of \( m \) parallel business processes \( B_{P_m}(R_1, R_2, ..., R_m) \) which starts at system time \( S_0 \), the completion of a workflow activity \( a_i \) contributes to the completion of the entire batch of processes with a value of \( M(a_i)/T \) where 
\[
T = \sum_{i=1}^{m} M(R_i) .
\]
Here, assume at the current system time point \( S_t \), the set of new completed activities from the last nearest system time point \( S_{t-1} \) (i.e. \( S_t - S_{t-1} = t_b \) ) is denoted as \( a_i \{S_t\}_{S_{t-1}} \), then the current system throughput is defined as 
\[
TH(S_t) = M(a_i\{S_t\}_{S_{t-1}})/T .
\]

Definition 2 (Candidate Throughput Checkpoint). Given the same batch of processes in Definition 1, a system time point \( S_t \) is a candidate throughput checkpoint if \( S_t - S_0 = n \times t_b \) (\( n = 1,2,3,\ldots \)). The checkpoint selection strategy will determine whether \( S_t \) is a throughput checkpoint or not.

Definition 3 (Throughput Consistency Model). Given the same batch of processes in Definition 1, and its final deadline denoted as \( F(B_{P_m}) \), then at a throughput checkpoint \( S_p \), the throughput consistency state of \( B_{P_m} \) is said to be: consistent if 
\[
R(a_i\{S_p\}) + \frac{M(a_i\{F(B_{P_m})\})}{m \times F(B_{P_m}) - S_p} \leq \frac{M(a_i\{F(B_{P_m})\})}{m \times F(B_{P_m}) - S_0} .
\]
Otherwise, inconsistent, i.e. a throughput violation is detected.

With the above new definitions, our novel time-point based checkpoint selection strategy is presented as follows.

**CSSTP**: Time-Point Based Checkpoint Selection Strategy.

At a system time point \( S_p \), if 
\[
R(a_i\{S_{p-1}\}) > \frac{M(a_i\{R(B_{P_m})\})}{m \times F(B_{P_m}) - S_p} \text{, then } S_p \text{ is selected as a throughput checkpoint; Otherwise, it is not selected as a throughput checkpoint.}
\]

Here, we describe a typical temporal checkpoint selection and verification process. At runtime, when it comes to a candidate throughput checkpoint, **CSSTP** will determine whether it should be selected as a throughput checkpoint or not. If not, nothing happens. If yes, the current throughput consistency state will be evaluated according to the throughput consistency model. If it is consistent, nothing happens. If it is inconsistent, an exception handling strategy will be triggered to recover the detected throughput violation.

The overhead for our strategy which mainly consists of the overhead for data query and the overhead for computation is very small. It is very close to the overhead for the conventional activity-point based strategy since no additional data is required and the computation only requires simple calculations. The major data required such as the mean completion time is static data which can be pre-fetched and the runtime completion time is dynamically recorded by the system usually with a logging component. Therefore, there is no additional overhead for data collection and trivial overhead for data query.

3. EVALUATION

The simulation experiments are conducted in our SwinDeW-C cloud workflow system [6]. The experiment simulates a batch of 1000 business processes running in parallel, each with 20 activities, and a total of 10 batches. The activity durations are randomly generated and deliberately extended by a mixture of representative distribution models such as normal, uniform and exponential to reflect the performance of different cloud services. The mean activity durations are randomly generated in a wide range of 30 milliseconds to 30 seconds. Meanwhile, some noises are also added to a randomly selected activity in each business process to simulate the effect of system uncertainties such as network congestion and performance down time. Different ratio of noises (the added time delays divided by the activity durations) from 5% to 30% are implemented. The strategy for setting the final deadline is similar to the work in [5] where a normal percentile is used to specify temporal constraints and denotes the expected probability for on-time completion. Here, we specify the normal percentiles as 1.28 which denotes the probability of 90% for on-time completion if without any temporal violation handling. This setting can be regarded as the norm, i.e. the satisfactory performance for most clients and service providers. We employ the state-of-the-art activity-point based checkpoint selection strategy **CSSTPD** introduced in [2] as benchmark. The basic time unit for monitoring is set as 60 seconds. For comparison purpose, we record the violation rates for the final deadline under natural situations, i.e. without any handling strategies (denoted as NIL). Meanwhile, we implement the simple elastic resource provision strategy (denoted as SERP) as the exception handling strategy. SERP employs one additional service instance when a temporal violation (either a response time violation detected by **CSSTPD** or a throughput violation detected by **CSSTP**) is detected.

**TABLE 1. Numbers of Temporal Violations**

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Normal</th>
<th>Uniform</th>
<th>Exponential</th>
<th>Noise</th>
<th>Number of Checkpoints by CSSTP</th>
<th>Number of Checkpoints by CSSTPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30%</td>
<td>50%</td>
<td>10%</td>
<td>5%</td>
<td>5158</td>
<td>282</td>
</tr>
<tr>
<td>2</td>
<td>30%</td>
<td>50%</td>
<td>10%</td>
<td>5%</td>
<td>5108</td>
<td>282</td>
</tr>
<tr>
<td>3</td>
<td>40%</td>
<td>50%</td>
<td>10%</td>
<td>10%</td>
<td>5322</td>
<td>473</td>
</tr>
<tr>
<td>4</td>
<td>40%</td>
<td>50%</td>
<td>10%</td>
<td>15%</td>
<td>5601</td>
<td>690</td>
</tr>
<tr>
<td>5</td>
<td>60%</td>
<td>40%</td>
<td>10%</td>
<td>20%</td>
<td>5466</td>
<td>671</td>
</tr>
<tr>
<td>6</td>
<td>60%</td>
<td>40%</td>
<td>10%</td>
<td>20%</td>
<td>5715</td>
<td>671</td>
</tr>
<tr>
<td>7</td>
<td>80%</td>
<td>40%</td>
<td>20%</td>
<td>20%</td>
<td>5877</td>
<td>888</td>
</tr>
<tr>
<td>8</td>
<td>80%</td>
<td>50%</td>
<td>20%</td>
<td>25%</td>
<td>5719</td>
<td>1057</td>
</tr>
<tr>
<td>9</td>
<td>80%</td>
<td>50%</td>
<td>20%</td>
<td>25%</td>
<td>5719</td>
<td>1057</td>
</tr>
<tr>
<td>10</td>
<td>30%</td>
<td>50%</td>
<td>20%</td>
<td>30%</td>
<td>5895</td>
<td>1257</td>
</tr>
</tbody>
</table>

Table 1 shows the number of checkpoints in each round of experiment. Clearly, the number of checkpoints selected by **CSSTPD**, i.e. the total number of response time violations for individual processes, is much higher than the number of throughput checkpoints selected by **CSSTP**, i.e. the number of throughput violations for the parallel processes. The reduction rate of the checkpoints ranges from 98% to 78% with an average of 87%. The results are very impressive where on average an order of magnitude reduction has been achieved. Meanwhile, the number of response time violations and the number of throughput violations both grows rapidly with the increase of noise. In contrast, the distribution of service performance seems to have less effect on them. For example, given the same noise, the average difference of throughput violations (e.g. R4 and R5, R6 and R7, R8 and R9) is around 20.
more checkpoints, namely exception handling cost. As for
However, as shown in Table 1, this is achieved by nearly 10 times
all detected response time violations of individual processes.

Temporal checkpoint selection plays an important role in
achieving the on-time completion of business processes. However,
conventional activity-point based checkpoint selection strategy
will require an extreme large number of checkpoints in order to
monitor a batch of parallel processes, and thus could result in a
violation rate ranges from 37% to 84% with an average of 65%. In
contrast, with either CSSTD or CSSTP , when either a response
time or a throughput violation is detected, the SERP strategy will
automatically employ one additional service instance to speed up
the execution of the business processes. As shown in Figure 2.
CSSTD can maintain a very close to 0% violation rate by handling
all detected response time violations of individual processes.
However, as shown in Table 1, this is achieved by nearly 10 times
more checkpoints, namely exception handling cost. As for CSSTP,
the temporal violation rates range from 1% to 8% with an average
around 4% which satisfies the 90.0% on-time completion rate
required by the settings of the deadlines. This result is very
impressive and promising since on average an order of magnitude
cost reduction has been achieved while the increase in the
violation rate is very small.

4. CONCLUSIONS AND FUTURE WORK

Temporal checkpoint selection plays an important role in
achieving the on-time completion of business processes. However,
conventional activity-point based checkpoint selection strategy
will require an extreme large number of checkpoints in order to
monitor a batch of parallel processes, and thus could result in a
huge monitoring and exception handling cost. To address such an
issue, this paper proposed a brand new idea which selects system
time points as checkpoints to simultaneously monitor the batch of
parallel processes using throughput as the performance indicator.
A set of new definitions including the throughput, the candidate
throughput checkpoint, the throughput consistency model, and our
novel time-point based checkpoint selection strategy CSSTP have
been proposed. The initial experimental results show that our
strategy can achieve an order of magnitude reduction in the
number of checkpoints compared with the state-of-the-art activity-
point based checkpoint selection strategy while also maintain a
satisfactory on-time completion rate.

With the best of our knowledge, this is the first paper that
proposed the novel idea of time-point based checkpoint selection.
The initial results presented in this paper promise a new research
direction in the area of software temporal verification which is
worth further investigation.

As mentioned in Section 2, since this paper is mainly for
presenting the novel idea and some preliminary results, we tried to
simplify the data, models and methods used in this paper as much
as possible and leave the complex extensions as future work.

Specifically, there are at least three research topics we need to
address in the near future:

1) To further reduce the violation rate. The handling of throughput
violations is no doubt a complicated issue. The SERP strategy
employed is very intuitive and mainly for the evaluation purpose.
In the future, more effective exception handing strategies will be
developed and thus may further reduce the violation rate.

2) To further improve the throughput consistency model. The
current throughput consistency model is very intuitive which only
defines two states. As studied in [5], the probability based
consistency models can be defined so that fine-grained temporal
violations can be detected and handled accordingly by different
exception handling strategies to save the handling cost.

3) To further improve CSSTP. “Necessity and Sufficiency” is the
ultimate goal for any generic checkpoint selection strategy. The
current method used in CSSTP which only compares the runtime
throughput with the mean throughput at a specific time point is
very intuitive. It may turn out that after a checkpoint is selected,
no throughput violation is detected, i.e. not necessary. Meanwhile,
since the experimental results showed that the global violation rate
is higher than 0%, it may also not be sufficient. In the future, more
effective methods can be designed to improve the performance of
CSSTP and ideally to achieve both necessity and sufficiency.

5. ACKNOWLEDGMENTS

The research work reported in this paper is partly supported by
Australian Research Council under Linkage Project LP0990393.

6. REFERENCES

Selection for Dynamic Verification of Fixed-time Constraints in
Grid Workflow Systems, Proc. 30th International Conference on

Selection for Dynamic Verification of Temporal Constraints in
Scientific Workflow Systems, ACM Transactions on Software
Engineering and Methodology, vol. 20, Article 9, 2011.

Workflow Systems,” Proc. 11th International Conference on
Advanced Information Systems Engineering, Heidelberg, Germany,
1999, pp. 286-300.

Computing: From Parallel Processing to the Internet of Things:
Morgan Kaufmann, 2012.

[5] X. Liu, Y. Yang, Y. Jiang, and J. Chen, Preventing Temporal


of Temporal Constraints in Production Workflows, Knowledge and

Proc. Workshop on Software Engineering for Cloud Computing, in

Mechanism for Service-Based Applications in the Cloud," IEEE