Measuring the Impact of Suspension on the Process Enactment Environment during Process Evolution

Pieter Hens[1], Monique Snoeck[1], and Manu De Backer[1,2,3]

[1] KU Leuven, Dept. of Decision Sciences and Information Management, Naamsestraat 69, 3000 Leuven, Belgium
[2] Universiteit Antwerpen, Dept. of Management Information Systems, Prinsstraat 13, 2000 Antwerpen, Belgium
[3] Hogeschool Gent, Dept. of Management and Informatics, Kortrijksesteenweg 14, 9000 Gent, Belgium

Abstract. Current workflow management systems implement the ability to automatically execute predefined process models. However, processes change over time and therefore a redeployment process has to be implemented to propagate changes into the running process enactment environment. One of the necessary steps in change propagation is to suspend the current process execution. This suspension does however decrease availability of the workflow management system, increases downtime and implicitly also decreases scalability. In this paper we provide a quantification of the impact of suspension on the runtime process enactment environment and experimentally evaluate this impact, hereby providing a better insight in suspension impact. Furthermore two suspension techniques are compared and a discussion is provided in which situations, which suspension technique is beneficial.

Key words: Workflow Enactment, Process Evolution, High Availability Systems

1 Introduction

Workflow management systems allow for the automated coordination and support of business processes. A business process represents the organizational flow of control and information from one process entity to another. By using an executable business process modeling language to describe the process (like BPMN2.0 [1] or BPEL [2]), a *process engine* can be used to read, interpret, deploy and execute the predefined process model [3]. However, due to changing business requirements, unexpected situations, changing environmental conditions, etc., the system should support the continuous restructuring and redeployment of the deployed process model. A current challenge for any workflow management system is to be able to respond effectively to these process changes.

Two types of runtime process changes can be identified: *ad-hoc change* and *evolutionary change* or *process evolution* [4]. The former handles case or instance specific changes and the latter handles case-independent, process model restructuring. In the rest of this paper we restrict ourselves to evolutionary changes. Process evolution means that over time the currently running process model is changed and this modified model has to be deployed in the runtime architecture. Process evolution is typically supported in a workflow management system by adopting a versioning mechanism. As shown in figure 1, each process instance is linked to its corresponding process model (schema) version. This allows new process model versions to be deployed over time. After (re-)deployment, each new process instantiation request is handled with the newest (most current) schema version. In order to also support process changes which not only affect new, but also already running process instances, it is required to migrate running process instances to the new process schema version. When deploying a new schema version, it is determined, according to a specific correctness criterion [5], which running process instances can migrate to the newest version and which can not. Migration is done by simple relinking the process instance and schema version (see figure 1), hereby propagating the change to the running process instances. In more detail, each deployment of a new process schema version undergoes the following steps:

1. Suspend the enactment of all process instances;
2. Determine which process instances can migrate;
3. Deploy the new version and migrate the identified process instances; and
4. Resume the process enactment

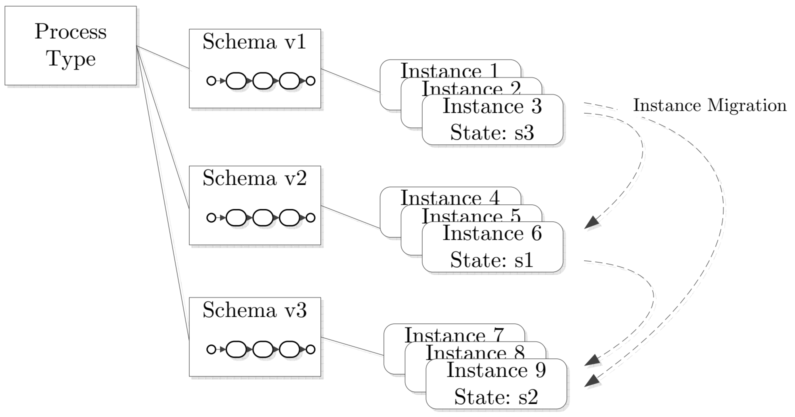

Fig. 1. Versioning and instance migration in a workflow management system

Suspension is necessary, because state based knowledge is required to determine if a process instance can migrate. Globally, instance migration can happen when the current execution state of the process instance is compliant with the new process schema. If the process instance is not suspended, the execution state of the instance can change during the deployment process, resulting in a possibly inconsistent system's execution state (i.e. instances which are not able to migrate are migrated nonetheless). A first objective of this paper is to examine how this suspension affects the process enactment environment during process evolution. Especially if lots of process instances have to be inspected to determine process instance migration, process enactment suspension creates a possibly unwanted downtime of the system during process evolution. This downtime could be problematic for systems which need to be highly available. For example, the foreign exchange market is a continuous market which accepts orders 24 hours a day. The processes behind these trading systems therefore have to be highly available and downtime should be limited to a minimum. Because of the necessary suspension, changing a process model and deploying it in the runtime environment disrupts the process execution, and therefore the business (e.g. trading). In which way the suspension during process evolution disrupts the enactment environment is however not yet investigated. This paper provides a better insight into the impact of suspension on the process enactment environment. The focus of this research is on process evolution for high availability systems.

Because suspension during process evolution can be harmful to some systems, we also investigate if a different approach to process evolution can be beneficial for those systems. Typically, process instances are suspended in their entirety, no matter the execution state of the instance. We call this *global suspension* (see the example in the top part of figure 2). Global suspension means that the process engine controlling the process instances is suspended, therefore suspending every running process instance and halting any state change in those instances. In a previous project we introduced a technique which enables, in contrast to global suspension, *partial suspension* of a process enactment [6]. Partial suspension allows the suspension of only specific parts of the process model, where the rest of the process can keep executing. The process engine is therefore not suspended in its entirety, but is still allowed to control specific parts of the process model. The bottom part of figure 2 shows an example of partial suspension. This suspension technique has the obvious advantage that during process evolution, some parts of the process model are not suspended and therefore do not experience downtime. A second goal of this paper is therefore to investigate how the reduced downtime resulting from the partial suspension technique impacts the enactment environment and how this compares to the impact when using global suspension. Moreover we discuss if using a different approach to suspension during process evolution can actually improve the availability and scalability of a high availability workflow management system. To the best of our knowledge this is the first research which evaluates a suspension technique for process enactment.

We can formulate our research questions as follows:

RQ(1) What is the impact of suspension on the runtime processes during process evolution?
- RQ(a) What is the impact of global suspension on the runtime processes during process evolution?
- RQ(b) What is the impact of partial suspension on the runtime processes during process evolution?

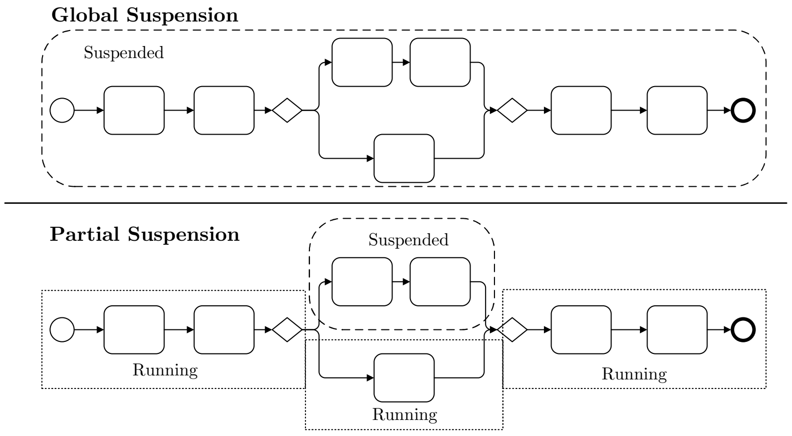

Fig. 2. Global versus partial suspension of the process enactment

RQ(2) Does a high availability process system benefit from using a different technique for process evolution (e.g. partial suspension)?

The contributions of this paper include the definition of four variables which measure the impact of suspension on the process enactment environment (section 4) and a provision of a better insight into the impact of suspension on the process enactment environment, varying over different system variables (section 4.1, 4.2 and 4.3). First, some background is provided in instance migration and the partial suspension technique (section 2). The paper concludes with a discussion of the benefits and disadvantages of partial and global suspension (section 5).

2 Background

In the experiments that investigate the impact of suspension on the enactment environment, both global and partial suspension is used. This allows the comparison of traditional process evolution with another suspension technique. In order to perform the experiments, we need a criterion that determines which process instances are able to migrate and we need a test environment that allows to (globally and partially) suspend, resume and migrate processes.

To determine which process instance is able to migrate, a migration criterion needs to be employed. For this purpose we used the change region technique introduced by van der Aalst [7]. The change region defines a region of states in the process model, computed from the old and new schema version. Figure 3 shows an example process model with two versions and their change region. A process instance with an execution state residing inside the change region is not able to migrate. If it resides outside the region, it is able to migrate. The deployment manager therefore checks the execution state of each (suspended) process instance and migrates any instance accordingly.

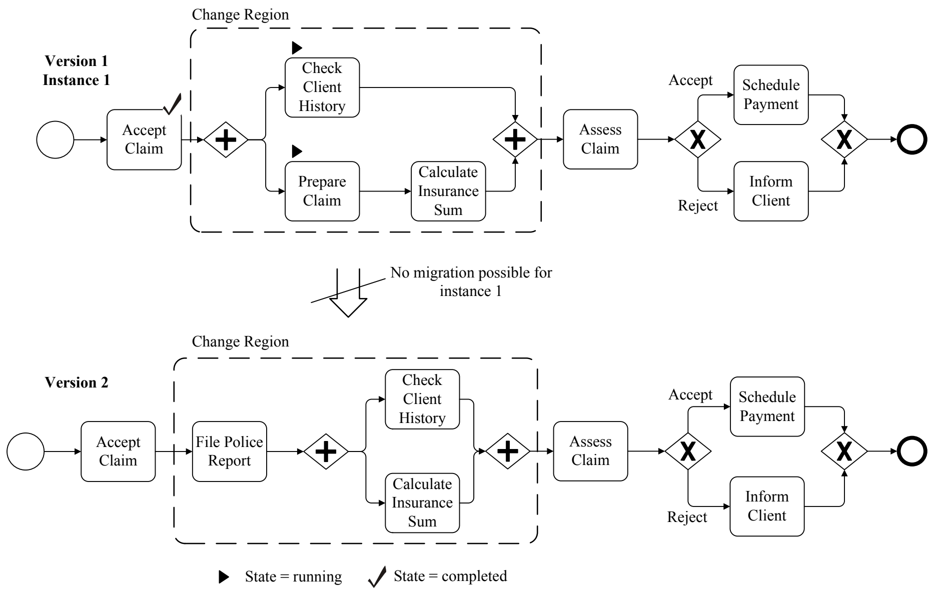

Fig. 3. Two versions of an insurance claim handling process

To implement partial suspension, a different method for process enactment has to be used. In a previous project we developed an approach to fragmented process enactment [8]. A process model is fragmented into logically different pieces, where each fragment is executed on its own, dedicated process engine (see figure 4). This fragmentation facilitates the fine grained control of the process model enactment. Each fragment's execution can be suspended, resumed, inspected and changed independently of others. For process evolution, this means that only a (sufficient) part of the process model execution can be suspended during model redeployment, in stead of the model in its entirety. In [6], we propose a process evolution protocol for the fragmented enactment environment and provide a method to determine the minimal part of the process model which needs to be suspended during process evolution. For the example in figure 3 the minimal fragments (tasks) that need to be suspended are the tasks *check client history*, *prepare claim*, *calculate insurance sum* and *accept claim*. All other parts of the model are still able to execute during redeployment, i.e. a process instance which reached an execution state not belonging to the suspended fragments can keep executing. For figure 3 this means that a process instance that resides in a state where task *assess claim* is being executed is still able to move to an execution state where *schedule payment* or *inform client* is triggered (the execution of this instance is not halted during process evolution).

Note that when a process instance is suspended, *process control* is suspended, not the actual work items (tasks) in the process. Already started tasks can finish their execution, regardless of the state of the process model enactment. Suspension therefore means that control (token) transfer in the process enactment is

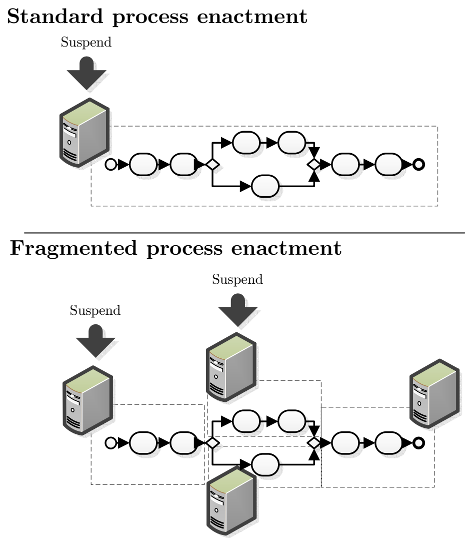

Fig. 4. Traditional and fragmented process enactment

halted (i.e. no new task executions are triggered), but already existing task instances can keep running. For example, a task "boat shipment" is not suspended during process evolution, but the state change in the process instance indicating the completion of the shipment task is not permitted.

For a further elaboration of the fragmented execution and the change region evolution protocol we refer to [6].

3 Experimental Setup

To analyze the impact of *partial (but sufficient) suspension* and *global suspension* on the enactment environment during process evolution, experiments are performed with our own implemented execution environment[1]. The effect of global and partial suspension is analyzed, varying over: process instantiation rate (section 4.1), task execution time (section 4.2) and position of the change in the process model (section 4.3). For each configuration, the throughput (process instance completions per time unit) is measured. To exclude randomness (from the Java program execution, CPU scheduling, etc.) each experimental configuration is run 3 times. The average throughput of these 3 runs is used for the further analysis. In the experimental setup we assume unlimited resources[2], to exclude

[1] The execution environment is implemented in Java and run on an Intel Core 2 Duo, 2.66GHz, with 4GB RAM configuration.

[2] This is simulated by not letting tasks perform any actual work (e.g. no SOAP call to a web service), but *'sleep'* through their designated task time. The execution environment itself has a sufficient performance to handle a large amount of simultaneous process instances and therefore does not interfere with the measurements [8].

performance issues of process coordination [8]. This way we only measure the influence of the suspension technique on the process execution. In all but one experiments the two versions of the insurance claim handling process model as shown in figure 3 is used as experimental model (since this model is also used in earlier work [6]).

Throughput of the process enactment is measured for a certain time period, in which the following two steps are performed:

1. Deploy version 1 of the insurance claim handling process; and
2. At a fixed time ($t = 6$), version 2 of the insurance claim handling process is deployed into the running enactment environment. This triggers the process evolution steps as described in the introduction. The change region and suspension criteria are calculated; the process enactment is suspended; the new model is deployed; any possible running process instances are migrated and the enactment is resumed again.

Note that, since we focus on high availability and continuous systems, the (independent) clients continue sending process instantiation requests during the process evolution deployment process (at a specified request rate).

4 Analysis

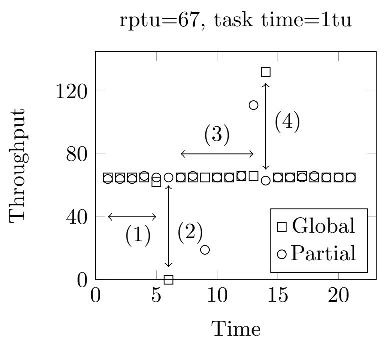

Fig. 5. Throughput over time for a process execution where a process change happens at time 6

Figure 5 shows the throughput measured over time for both global and partial suspension during process evolution. Time is indicated in units (*tu*), throughput is measured per time unit and the client sends a constant number of requests per time unit (*rptu*). From the graph we can identify four disparity variables, quantifying the impact of suspension on the process enactment environment:

(1) suspension-start indicating when the effect of suspension is noticeable in throughput;

(2) suspension-base indicating the lowest throughput during process evolution;
(3) catchup-time indicating the time needed to handle any backlog buildup and revert back to the standard (average) throughput; and
(4) backlog-summit indicating the highest throughput during process evolution, i.e. the backlog that formed during suspension.

In figure 5 we see that the suspension-base with global suspension equals zero. This will always be the case, since the entire process execution is suspended, thus not allowing any process instances to complete over a certain time period. For partial suspension, the suspension-base never reaches zero, as the environment is still able to complete some process instances during suspension (since not every fragment is suspended, see section 2). Similarly, the backlog that is created during suspension is noticeably smaller for partial suspension. Another prominent difference is that the effect of suspension is only visible at a later time for partial suspension than for global suspension. The catchup-time is therefore also smaller for partial suspension.

In the next sections, we vary over different configurations, investigating how these four variables fluctuate, comparing partial and global suspension. Since in some configurations, the throughput is much more irregular than the example shown in figure 5, we capture the four variables (dispersion of the throughput during process evolution) by means of the *Mean Absolute Deviation (MAD)*[3]:

$$D = \frac{1}{n}\sum_{i=1}^{n} |x_i - m(X)| \tag{1}$$

where $m(X)$ equals the average throughput under normal conditions (standard execution, no suspension and process evolution). The MAD is calculated for the entire time period of the experiment (20tu in this case). In the case of figure 5, $D(global) = 6.75$ and $D(partial) = 4.81$. A higher value means a bigger dispersion, and therefore a bigger throughput irregularity during process evolution.

4.1 Effect of RPTU on Suspension Throughput

To test the effect of varying process instantiation rate (or *system load*) on the throughput during process evolution, an experiment is set up that enacts the insurance claim handling process and where a change as described in section 3 is initiated. Throughput is measured for each time interval and the experiment is rerun for different process instantiation rates.

Figure 6 shows the depth of the suspension (suspension-base) for each request rate, for partial suspension as well as for global suspension. As already

[3] Note that MAD is closely related to standard deviation, also measuring dispersion. We have chosen MAD, because it gives a more comprehensible (descriptive) dispersion score. MAD can however not be used in mathematical statistics, which is no problem because we only use the dispersion scores for a relative comparison. For a discussion on MAD we refer to [9]

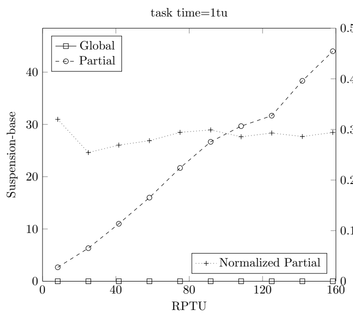

Fig. 6. Suspension-base with varying request rate

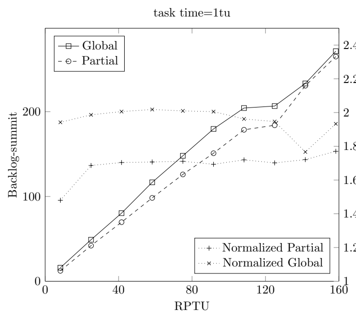

Fig. 7. Backlog-summit with varying request rate

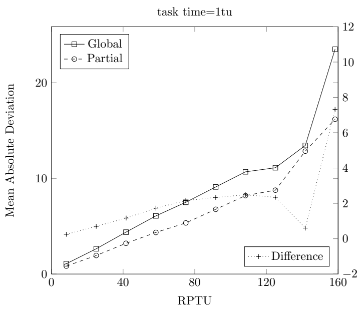

Fig. 8. Mean Absolute Deviation of throughput with varying RPM and constant task time

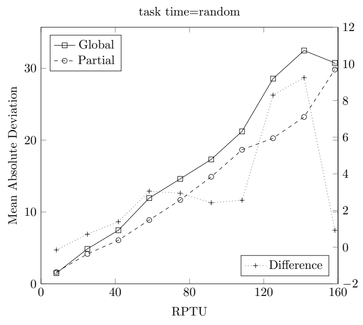

Fig. 9. Mean Absolute Deviation of throughput with varying RPM and random task time

mentioned, the suspension-base for global suspension is always zero, no matter the request rate. For partial suspension, we notice an increase in the minimum throughput during process evolution as the request rate increases. This is a consequence of the increased process instantiation rate. Since more requests are sent per time unit, more instances can also be completed per time unit. Under the assumption of unlimited resources (see section 3), the suspension-base increases linearly with respect to request rate. For limited resources, throughput will reach a maximum at a certain load (request rate) and the suspension-base will flatten out at higher request rates. See [8] and [10] for process execution performance tests.

To evaluate if rptu has a relative effect on the suspension-base, the normalized values of the suspension-base for partial suspension are also depicted on figure 6. From the normalized values we can conclude that the suspension-base does not increase or decrease, relative to the request rate. The same conclusions can be drawn for the backlog-summit (see figure 7). At higher request rates, the backlog will be bigger (since more process instantiations are requested during suspension). However, relative to the request rate, the backlog-summit stays constant for either partial and global suspension.

Figure 8 shows the MAD for partial and global suspension. Here we can conclude that partial suspension has a slight advantage over global suspension, i.e. the process throughput is less irregular during process evolution, using the partial suspension technique. This advantage is however not dependent on request rate. In general, the throughput during process evolution is fairly resilient against varying request rates for both partial and global suspension techniques.

The previous results are all experiments performed where each task has a similar, constant task time. This is not realistic as every task execution can have a different duration (even within the same process instance), which is especially true for human tasks. To inspect the influence of different task times, figure 9 shows the MAD for the same experiment, but with random task times (random per task, per instance). Although the MAD is higher, the conclusions are similar to the experiments with constant task times. There is no influence of request rate on throughput during process evolution and partial suspension undergoes a smaller throughput irregularity than global suspension. Further experiments are therefore done with a constant task time, so that any randomness will not interfere with the results.

Suspension-start and catchup-times are in this case not reported because they stay constant at any request rate.

4.2 Effect of Task Times on Suspension Throughput

In this section we evaluate the effect of task execution times on the throughput during process evolution. An experiment is set up with the same configuration as in the previous section with the difference that we vary over task times and not process instantiation rate (rptu=70).

Figure 10 and 11 show the throughput during process enactment and process evolution, for a task timing of $\frac{1}{5}$tu and a timing of 2.6tu. The big difference between the two throughput measurements is the much higher dispersion when the task time is bigger (Figure 11). In stead of one drop in throughput and one increase to handle the backlog, there is an entire range of drops and increases until the process enactment stabilizes again (catchup). This is especially apparent for the global suspension. As the process gets suspended, the throughput drops to zero at time t, at time $t+1$ the throughput increases to $\pm$ the backlog-summit and drops again to zero at time $t+2$. This alternation continues until the enactment stabilizes. Figure 12 explains why this phenomenon happens. During suspension,

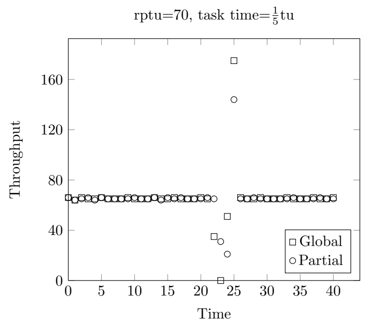

Fig. 10. Throughput over time, with task execution time=$\frac{1}{5}$tu

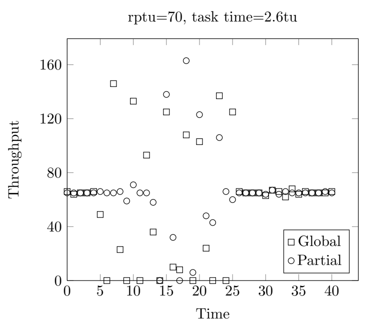

Fig. 11. Throughput over time, with task execution time=2.6tu

each task is still able to finish its execution, only the process engine is suspended (see section 2). This means that, during suspension, a backlog of control flow tokens which need to be transfered to the next task in the process model is created for each task in the model (see figure 12). Once the process enactment is resumed, the backlog is transfered in its entirety to the next task in the process flow (each token buffer is shifted one place to the right). Hereafter, each task will handle the received backlog in its entirety and simultaneously (process instances are handled in parallel). Since task execution takes a significant amount of time ($2.6tu$), throughput drops to zero in the first time period after resumption (no process instances are completing, since there are still tasks left to be done). When the tasks finish execution of the backlog, the token buffer is again shifted one place to the right in the process model. For the last task in the model, this means that the respective process instances are completed. A higher throughput is therefore measured (all instances in the last task's backlog finish simultaneously). This alternation continues until every created backlog per task is handled.

Figure 13 shows the MAD with varying task times and an rptu of 70. The absolute difference between global suspension and partial suspension is also depicted. From the incline of the difference graph we see that there is a larger effect of increasing task times on throughput irregularity for global suspension than partial suspension. The larger the task timings, the larger the relative benefit of using partial suspension as a suspension technique in process evolution.

4.3 Effect of the Change Position on Suspension Throughput

To measure the effect of the change position in the process model, we used the test model shown in figure 14. The change propagated in the enactment system is a task deletion, resulting in the partial suspension of two tasks, the deleted

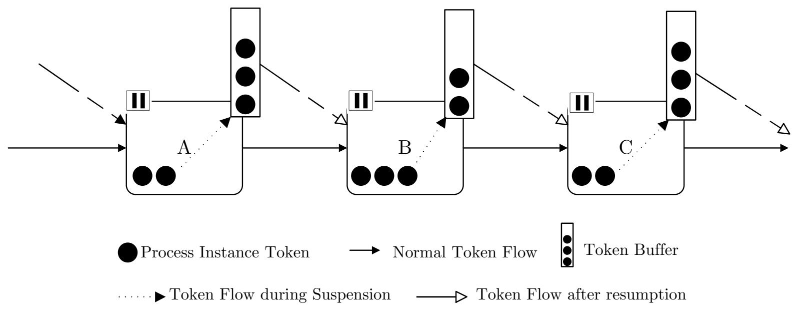

Fig. 12. The reason for alternating suspension-base/summit with larger task execution times

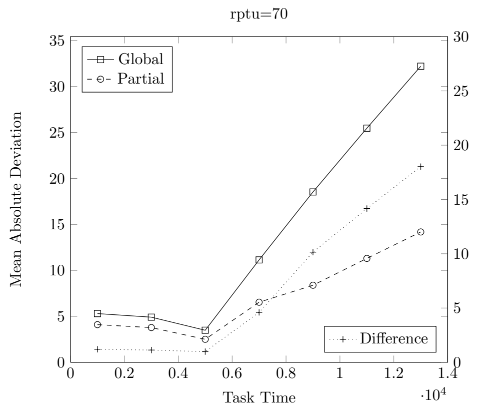

Fig. 13. Mean Absolute Deviation of throughput with varying task time

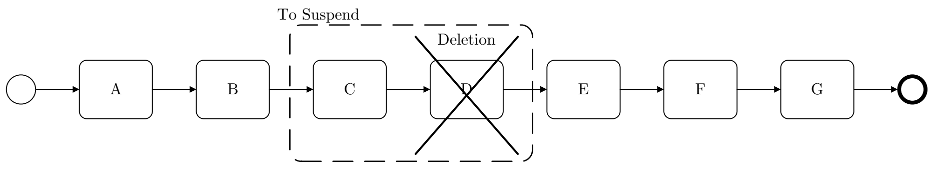

Fig. 14. Test model for measuring the effect of the change position on suspension throughput

task and the task in the upward flow (see section 2). Throughput is measured for each change propagation: deletion of task A, deletion of task B, etc.

Figure 15 shows the throughput when a change happens in the beginning of the process model (task A is deleted) and figure 16 shows the throughput when a change happens at the end of the process model (task G is deleted). The difference in suspension-start for the partial suspension technique is clearly noticeable from these two graphs. When a change happens in the beginning of the process flow, the effect of the suspension is visible in throughput at a later time than when the change happens at the end of the process flow. Figure 17

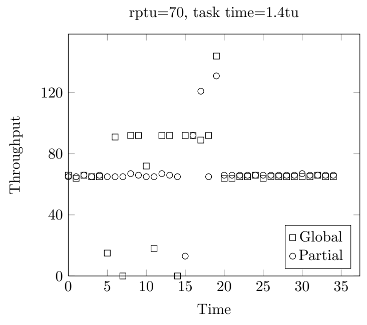

Fig. 15. Throughput over time, with a change at the beginning of the process flow (delete A)

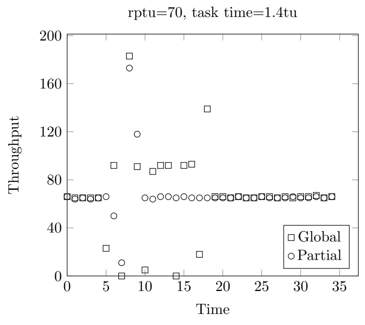

Fig. 16. Throughput over time, with a change at the end of the process flow (delete G)

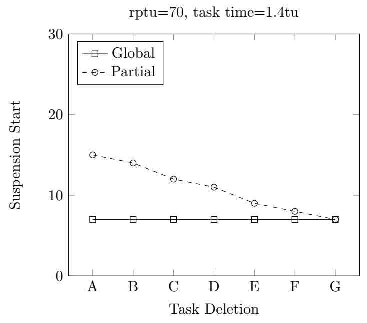

Fig. 17. Suspension-start with varying change position

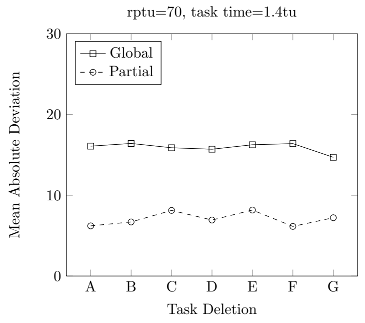

Fig. 18. Mean Absolute Deviation of throughput with varying change region position

provides an overview of suspension-start across varying change positions. The shifting suspension-start for partial suspension is due to the fact that when the suspension zone is at the beginning of the process model, every process instance already past the suspension zone can still complete without any downtime (the effect is only noticeable at a later time). When the suspension zone is at the end of the process model, almost every running process instance is *blocked* by the suspended fragments (the effect is immediately visible).

The MAD for each change position configuration is however always the same (figure 18). Suspension-start varies according to the change position, but the catch-up time, suspension-base and backlog-summit are constant.

5 Discussion

In the previous section we measured throughput during process evolution, quantifying the impact suspension has on process enactment. Comparing the results we see that with the partial suspension technique, throughput experiences a smaller irregularity than when using global suspension. Indeed, partial suspension leaves room for process instances to still complete and execute, even when the system is suspended. This decreases downtime of the overall process enactment environment. For increasing task times, partial suspension even has an increasing relative advantage over global suspension. In both the other cases, changing request rate and changing position, the relative advantage of partial suspension is constant (but existing nonetheless).

Besides reduced downtime, another advantage of partial suspension observed in the experiments is a diminished backlog buildup. Backlog buildup and an irregular execution can be problematic for less scalable systems. In a real-life scenario, resources are limited and if the backlog grows larger, more process instances have to be handled simultaneously. At heavy loads the performance of process execution degrades and throughput drops [10]. In [8] we show that already at 83rptu, a process engine significantly loses performance and has trouble catching up. A high backlog buildup will therefore decrease performance, further increasing the irregularity during process evolution.

There is thus an advantage of using the partial suspension technique, but it can be argued that it remains fairly small. The process evolution protocol is rather quick, which limits the downtime (suspension time) of the enactment. In our experiments, the catchup-time when using global suspension is $\pm$ 2 minutes (for 10 second task timings). At some point in time throughput equals zero for global suspension, but the downtime stays limited to a very small time period. The advantages of partial suspension are therefore limited to systems which satisfy the following assumptions: high availability is required, there is a continuous client request rate and process evolution is performed during this (busy) process enactment. In the introduction we already gave one example of such a system: the foreign exchange market, which satisfies all the assumptions. Other systems could satisfy the assumptions on specific (unforeseen) moments and benefit from the partial suspension technique in these situations. For example, in a Short Message Service (SMS) [11] process implemented by a telecommunications company, it is defined that after each SMS sent a confirmation of receipt is sent back to the sender of that SMS. During high loads of the network (e.g. big events, natural disasters [12], ...), the company may want to disable the confirmation SMS, and therefore has a need to change the implemented process during process enactment. Since availability is important in these situations, the downtime caused by process evolution should be limited. In this specific case is partial suspension during process evolution beneficial.

The benefits of partial suspension are therefore only limited to very specific environments, but it can be very valuable on these systems.

6 Conclusion and Future Research

In this paper we investigated the impact of suspension on the process enactment environment when process evolution is needed. A better insight in the effect of suspension is provided by defining four variables which quantify this effect: suspension-start, suspension-base, catchup-time and backlog-summit. During suspension, process throughput drops and a backlog is created which increases the load on the enactment system. Furthermore we compared two different suspension techniques for process evolution: global suspension and partial suspension. The partial suspension technique has a reduced impact on the process enactment environment, provides for a smaller downtime and decreases the backlog buildup. These benefits are however only profitable in very limited situations: high availability systems that have a continuous process instantiation request rate. For these systems, the partial suspension technique can be very valuable. In other cases the advantages of partial suspension do not outweigh the overhead of implementing the technique in the workflow management system.

This paper is focused on evolutionary changes. Future research involves investigating the impact of ad-hoc, case-specific changes. Is the change impact of updating a case-specific scenario similar to the impact of propagating a case-independent change?

References

1. Object Management Group: Bpmn 2.0. http://www.omg.org/cgi-bin/doc?dtc/10-06-04 (June 2010)
2. Oracle: Bpel process manager (May 2010) `http://www.oracle.com/technology/products/ias/bpel/index.html`.
3. Van der Aalst, W., ter Hofstede, A., Weske, M.: Business process management: A survey. Business Process Management (2003) 1019–1019
4. Weber, B., Sadiq, S., Reichert, M.: Beyond rigidity–dynamic process lifecycle support. Computer Science-Research and Development **23**(2) (2009) 47–65
5. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems—-a survey. Data & Knowledge Engineering **50**(1) (2004) 9–34
6. Hens, P., Snoeck, M., De Backer, M., Poels, G.: Process evolution in a distributed process execution environment. Submitted for International Journal on Information System Modeling and Design (2012)
7. van der Aalst, W.: Exterminating the dynamic change bug: A concrete approach to support workflow change. Information Systems Frontiers **3**(3) (2001) 297–317
8. Hens, P., Snoeck, M., De Backer, M., Poels, G.: An autonomous distributed system for business process execution. Submitted for Information Systems (2011)

9. Gorard, S.: Revisiting a 90-year-old debate: the advantages of the mean deviation. British Journal of Educational Studies **53**(4) (2005) 417–430
10. Nanda, M., Chandra, S., Sarkar, V.: Decentralizing execution of composite web services. ACM SIGPLAN Notices **39**(10) (2004) 170–187
11. Peersman, C., Cvetkovic, S., Griffiths, P., Spear, H.: The global system for mobile communications short message service. Personal Communications, IEEE **7**(3) (2000) 15–23
12. Ministery of Internal Affairs and Communications (Japan): Maintaining communications capabilities during major natural disasters and other emergency situations. `http://www.soumu.go.jp/main_content/000146938.pdf` (2011)