Int. J. Nonlinear Anal. Appl. 10 (2019) No. 1, 217-228
ISSN: 2008-6822 (electronic)
https://dx.doi.org/10.22075/IJNAA.2019.4065

Real-time Prediction and Synchronization of Business Process Instances using Data and Control Perspective

Iman Firouzian[a,*], Morteza Zahedi[a], Hamid Hassanpour[a]

[a]*Faculty of Computer Engineering and IT, Shahrood University of Technology, Shahrood, Iran*

(Communicated by M.B. Ghaemi)

Abstract

Nowadays, in a competitive and dynamic environment of businesses, organizations need to monitor, analyze and improve business processes with the use of Business Process Management Systems (BPMSs). Management, prediction and time control of events in BPMS is one of the major challenges of this area of research that has attracted lots of researchers. In this paper, we present a 4-phase pipeline approach to the problem of synchronizing each pair of dependent process instances to arrive at the corresponding pair of tasks simultaneous or near-simultaneous. In the first phase, the process model is mined from the event log and enriched by the probabilistic distributions of time information. In the second phase, the hidden processing dependency between the each pair of dependent process instances is formally defined and is mined from the event log. In the third phase, a process state prediction algorithm is presented to predict the future route of process instance and then predict the remaining time of the process instance to a given point in a predicted route of the business process. In the fourth phase, an iterative synchronization algorithm is presented based on the presented process state prediction algorithm to make each pair of dependent process instances arrive at the corresponding pair of tasks simultaneous or near-simultaneous. Experimental results on a real-life event log of BPI challenge 2012 show that the proposed method leads to 39% reduction in cycle time for dependent process instances.

Keywords: Business Process, Synchronization, Cycle Time Prediction, Dependent Process Instances.
2010 MSC: 68N30

*Corresponding Author

Email addresses: iman.firouzian@shahroodut.ac.ir (Iman Firouzian), zahedi@shahroodut.ac.ir (Morteza Zahedi), h.hassanpour@shahroodut.ac.ir (Hamid Hassanpour)

Received: December 2017 *Revised:* November 2018

1. Introduction

In many companies and organizations, information systems are facing the problem of handling enormous amount of data. Therefore, BPMS provides a framework to better manage the business processes [1-3]. BPMS tends to manage business processes more intelligently. While the older tools of workflow management systems (WFMS) provide development, execution and management of business processes, BPMS additionally provides the possibility to manage the interaction between processes and provides more compatibility for process models with reality [3, 4].
Handling task dependencies is one of major issues that affect the cycle time of business processes. In manufacturing environments, cycle time refers to the time duration between customer request and product delivery to the customer. Cycle time reduction in business processes such as manufacturing processes is an important optimization that can be achieved in the business process with an appropriate scheduling and planning [5-8].
In this paper, a subset of task dependencies is considered that state-of-the-art process mining algorithms are unable to discover. In this kind of dependency, a business process instance waits in a task for the other corresponding business process instance to reach to the dependent task. In this paper, a prediction-based method is proposed which orchestrates and synchronizes each pair of dependent process instances to arrive at the corresponding pair of tasks simultaneous or near-simultaneous. The proposed approach uses an iterative method to constantly synchronize the process instances at each node each process instance takes a step ahead.
This main contributions of this paper are as follows:

- Defining and discovering the inter-task dependencies from event logs by statistical analysis.
- Prediction of remaining time of business process instances to a specified target point in business process model.
- Dynamic synchronization of each pair of dependent process instances to arrive at the corresponding pair of tasks simultaneous or near-simultaneous.

The remainder of this paper is organized in 5 other sections. The related researches and papers in the task dependency domain are discussed in section 2 categorized based on their objective. Section 3 starts by prerequisite definitions in the domain and continued by clarifying task dependencies extraction from event logs. The proposed method is presented in detail in section 4, focusing on leveraging the tasks dependencies in the core of the proposed algorithm. The experiments and discussion are presented in section 5. Conclusions of the paper are made in the last section.

2. Related Works

The description of dependencies has been a focus in different research areas. Next to services [9] and activities, dependencies between classes or software modules [10], features (mostly in area of product line management) [11], and requirements [12] are captured and evaluated for different purposes such as dependency based service composition or optimizing the design of large software systems. The approaches for representing dependencies vary. In [10], the authors present the idea to use a Dependency Structure Matrix (DSM) to model dependencies where dependencies between entities are represented by a mark in a matrix. The automatically generated matrix is used to optimize product development processes. It highlights e.g. cyclic dependencies between entities, which thereupon can be removed. However, a DSM is not suitable to create a DM. It does not support the explicit modeling of dependency features such as symmetric and asymmetric dependencies. Also,

within a DSM one entity cannot have more than one dependency to exactly one other entity, since only one value for each entry of the matrix can be specified.
Wu et al. use the DAG Synchronization Constraint Language (DSCL) to model different types of dependencies [13]. Data and control dependencies are expressed by synchronization constraints such as happenBefore or happenTogether. However, due to the nature of expressing dependencies as synchronization constraints, this limits its applicability to certain use cases. For the handling of dependencies between services and their SLAs (first use case) this is not sufficient. In DSCL those can only be expressed in terms of the constraint happenTogether. Thus the modeling of those types is hindered and not intuitive. A common approach to capturing dependencies are dependency graphs, where nodes represent the entities which are dependent on each other and edges represent the dependencies between these entities. One example is the work by Zhou et al. [9], who developed an approach for the automatic discovery of dependencies based on semantic web service and business process descriptions. They use a dependency graph to capture control and data dependencies and create a minimal dependency graph presenting all dependencies. Dependency type information and specific properties of the dependency are missing. This DM is limited and does not allow the specification of typed dependencies or properties such as bi-directional, inverse, and disjoint dependencies.
Most of the early process modeling languages has an imperative style and uses control constructs or workflow patterns to specify the skeleton of a process [14]. Until very recently, the necessity of providing a declarative flow language for service scheduling has caught up its pace. [15] proposed an approach of using temporal logic to specify the synchronization constraints between different components. Temporal logic provides a richer syntax for describing and monitoring synchronization dependencies. But its non-determinism makes it impractical for generating a message-exchanging synchronization protocol for synchronization enforcement.
Another related area of research is rule-based service composition [16, 17]. Rules are defined to decide role assignment in process execution, message exchange, and flow constraints etc. Most business rules could be recast to dependencies defined in our framework and used as input for service scheduling engine. For example, the structure related rules in [16] could be recast either as control dependencies or data dependencies. These early work focused more on rule classification and process modeling. By comparison, our work identified those dependencies crucial to service scheduling and studied their interaction affect. Therefore, our approach can server as a rule-based scheduling engine and plug into their systems.
A traditional approach to handling dependencies implicitly uses extended transaction models [18, 19] that introduce new data manipulation semantics more sophisticated than serializability. Typical methods to implementing extended transaction models, e.g. Reflective Transaction Framework [20], extend algorithms used in database management systems such as concurrency control. In contrast, our research results address the synchronization needs of programs, workflows, and data manipulations in a uniform way.

3. Formal Definition of Process Model

To formally express the proposed method, it is essential to present an analytical model for the function of business processes and associated characteristics and parameters. An analytical model of processes is presented as tuple $p = \langle \varphi, T, F, R, U, \mu, \delta \rangle$ as the following:

1. (a) $\boldsymbol{\varphi}$ is the arrival rate of incoming requests.
   (b) set $\boldsymbol{T}$ is the set of given tasks in the business process.
   (c) set $\boldsymbol{F} \subseteq \boldsymbol{T} \times \boldsymbol{T}$ is a set of directed connections between tasks which represents the work flow of the given set of tasks of business process.

(d) set $\boldsymbol{R}$ contains all the human resources in organization by which the given tasks are done.
(e) $\boldsymbol{U} \subseteq \boldsymbol{T} \times \boldsymbol{R}$ defines the responsible resource for each task. As an example, ⟨r,t⟩∈U shows resource r in process p is responsible for task t.
(f) $\boldsymbol{\mu} : \boldsymbol{U} \rightarrow \mathbb{R}^{+}$is a function, which maps each item $\langle t, r\rangle \in U$ onto the execution time of task t which is performed by resource r.
(g) $\boldsymbol{\delta}$**:**$\boldsymbol{U}\times\boldsymbol{T}\rightarrow \mathbb{R}^{+}$ is a mapping from $U\times T$ to positive real numbers. Value $\delta(\langle t, r, t'\rangle)$ shows average dependent time**.**
(h) $\boldsymbol{D}$**:**$\boldsymbol{U}\rightarrow$**F** $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ is a mapping from set U to positive real numbers. Values μ, σ are parameters of probability distribution associated to processing time t and resource r abbreviated as $\mathrm{F}_{r,t}(\mu, \sigma)$.
(i) $\boldsymbol{Q} : \boldsymbol{R} \rightarrow \boldsymbol{T}^{*}$ is a function which associates a queue of work items to each resource in terms of sequences.

4. THE PROPOSED METHOD

In this section, a four-phase pipeline approach is proposed to the problem of synchronizing each pair of dependent process instances to arrive at the corresponding pair of tasks simultaneous or near-simultaneous. In the first phase, the process model is mined from the event log and enriched by the probabilistic distributions of time information. In the second phase, the hidden processing dependency between the each pair of dependent process instances is formally defined and is mined from the event log. In the third phase, a process state prediction algorithm is presented to predict the future route of process instance and then predict the remaining time of the process instance to a given point in a predicted route of the business process. In the fourth phase, an iterative synchronization algorithm is presented based on the presented process state prediction algorithm to make each pair of dependent process instances arrive at the corresponding pair of tasks simultaneous or near-simultaneous.

4.1. First Phase: Process Discovery

In this phase, the business process model is mined from the event log and enriched by the probabilistic distributions of time information. In the literature, algorithms such as alpha algorithm [21], heuristics miner [22], and genetic miner [23] are applied to automatically discover the business process lied in the event log. In this paper, alpha algorithm is chosen for discovering the business process from the event log. Since some modifications to alpha algorithm are further applied, the notations and the procedure of alpha algorithm are presented below. Suppose W is a workflow event log on T. $\propto(W)$ is defined as follows:

$$1.\ T_W = \{t \in T \mid \exists_{\sigma\,\epsilon\,W}\ t \in \sigma\ \}$$

$$2.\ T_I = \{t \in T \mid \exists_{\sigma\,\epsilon\,W}\ t = first(\sigma)\ \}$$

$$3.\ T_O = \{t \in T \mid \exists_{\sigma\,\epsilon\,W}\ t = last(\sigma)\ \}$$

$$4.\ X_W = \{\ (A, B) \mid A \subseteq T_W \wedge A \neq \emptyset \wedge B \subseteq T_W \wedge B \neq \emptyset \wedge$$
$$\forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a1,a2\ \in A} a_1 \#_W a_2 \wedge \forall_{b1,b2\ \in\ B}\ b_1 \#_W b_2\ \}$$

$$5.\ Y_W = \{(A,\ B)\ \in X \mid \forall_{(A',\ B')\ \in\ X} A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A,\ B) = (A', B')\}$$

In alpha algorithm, the relations between tasks are specified by four notations comprising $>$, $\rightarrow$, $\#$, $\|$. Suppose a and b are two tasks and $X \in (T - \{a,\ b\})^{+}$ is the trace of tasks omitting tasks a and b. The relations between tasks in alpha algorithm is defined as follows:

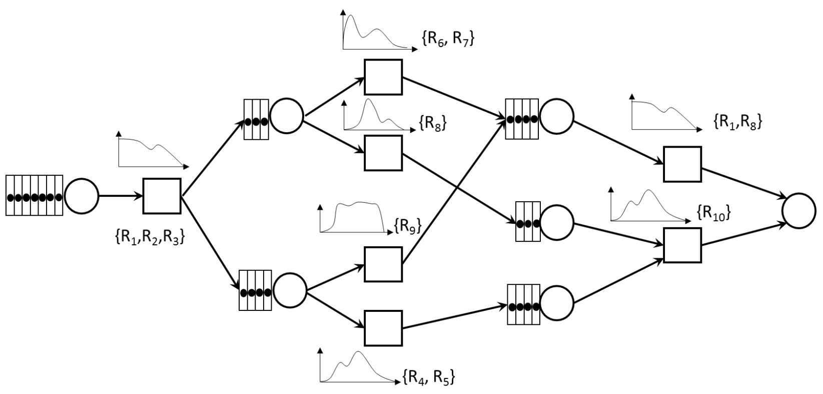

Figure 1: A process model enriched with probabilistic time information

1. *a*>*b* if task *a* is observed exactly before task *b* in some traces. In other words, string $XabX$ is a substring of some traces.
2. *a*→*b* if *a*>*b* and *b*≯*a*. In other words, string $XabX$ is observed in the event log but string $XbaX$ is not observed in any trace of the event log.
3. *a* # *b* if *a*≯*b* and *b*≯*a*. In other words, string $XabX$ or $XbaX$ is not observered in any trace of the event log.
4. *a*||*b* if *a*>*b* and *b*>*a*. In other words, both strings $XabX$ and $XbaX$ are observered.

After mining the process model from the event log using the above procedure, the process model is then enriched by the time information. Time information associated to each task is also present in the event log. Processing time of each task varies by factors such as the associated resource, the congestion of work items, the daytime of processing task and so on. Time information related to each task includes the activation time of a work item, waiting time of a work item in a queue, starting time of processing work item by associated resource, and ending time of processing work item by associated resource. The best can be done is to use all the prior knowledge and historical observations to create a probabilistic model for estimating the events in question.

4.2. Second Phase: Hidden Processing Dependency

It is common for process discovery algorithms to discover execution dependency between the tasks from event logs. Dependencies are categorized into two categories of local and non-local dependencies, but not all dependencies are discovered by common discovery algorithms such as alpha algorithm. In this paper, a special subset of non-local dependencies are considered in which processing a process instance in a task depends on processing another process instance in another corresponding task. In other words, a process instance waits for the dependent process instance to be processed. The increase in waiting time leads to increase in cycle time of all process instances. It is assumed that in synchronization, the two process instances could be instantiated from one business process or two different business processes. The hidden processing dependency is formally defined with an example as follows: Consider event log L as follows:

$$\begin{gathered} L = \{<A, E, B, F, G, C, D, H>^{70}, \\ < A, B, E, F, G, C, D, H>^{80}, \\ <A, E, F, B, G, C, D, H>^{70}, \\ <A, E, B, F, C, G, D, H>^{40}, \\ < A, B, E, F, C, D, G, H>^{80}\} \end{gathered}$$

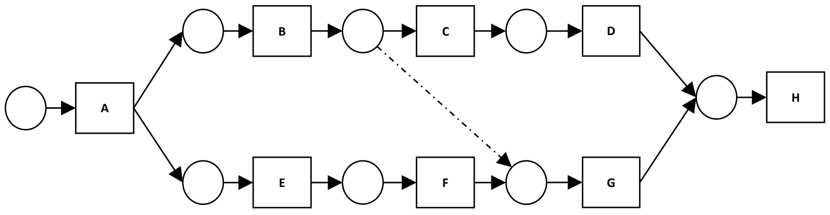

Figure 2: Hidden processing dependency shown by dotted connector

By investigating event log L, it is concluded that task B is always observed before task G, but task B is not always observed exactly before task G. In Fig. 8, tasks B and G are positioned in parallel route with a set of other tasks which are processed with any order in between the two tasks. The specific order for tasks B and G may imply to an implicit rule of temporal dependence by which notations of alpha algorithm are unable to describe. A new notation is defined for the scenario as follows:

> ***a* ::>*b*** if task a is always observed before task b in all traces of a given case and the processing order is not discovered directly or indirectly in the process model, the relation is considered a non-local dependency and is defined between tasks of one business process. If the relation c::>d does not hold for task c and d, the notation c::$\not>$$d$ is used.

By investigating the definition above in the event log related to process in Fig. Below, the relation B ::>G is established. In event log L, task B is always observed before task G and the order is not defined directly or indirectly in the process model. Therefore, hidden processing dependency between tasks B and G is defined. By the definition above, the relation A::$\not>$$H$ does not hold, because task A is indirectly placed before task H in the process model.

The definition presented for relation ::> is also applicable for two different business process scenario. If a case in a business process does have a corresponding case in another business process and there is a hidden processing dependency between them, the relation ::> could be defined between two tasks of two business processes of the event log. The hidden processing dependency could also be generalized to multi business process scenario. To better understand the inter-process dependency, consider the case when the owner of process instance instantiated from the first business process is the same as the owner of process instance instantiated from the second business process and the cases are corresponding to each other. It is important to discover the link of dependency between the two tasks from two different business processes.

In another process model depicted in the figure below, the processed token in task A is split into two tokens in tasks B and F. By mining event log using the proposed discovery technique, it is understood that processing task I depends on processing task C. In other words, a token in task I does not start in route FGHI until the corresponding token in task C finishes. Task D also depends on task H and a token in task D of route BCDE does not start processing until the corresponding token in task H of route FGHI ends.

4.3. Third Phase: Remaining Time Prediction

The route prediction of process instances requires investigating each branch ahead of the process model which would solve the ambiguity of multi-route problem. To this end, process instances are categorized in the place of each branching and then each process instance is assigned to a route. Support Vector Machines (SVMs) technique is used to classify the process instances in order to predict their future route. A SVM classifier is leveraged where tasks are faced with a route branching.

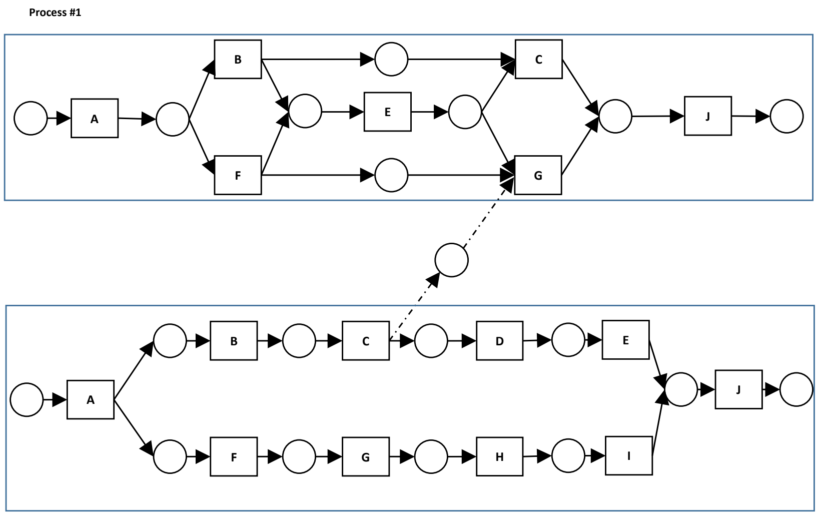

Figure 3: Hidden processing dependency between two different business processes

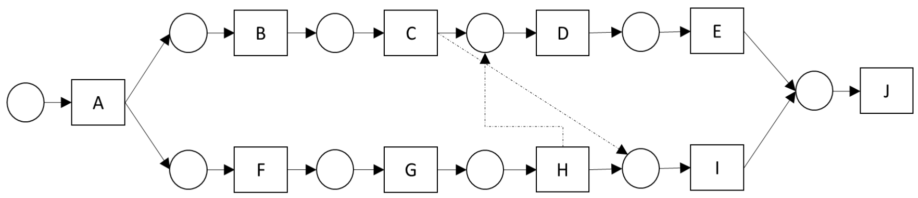

Figure 4: Hidden processing dependency in parallel route

Therefore, the required number of SVM classifiers in a process is equal to number of the tasks which are followed by route branching. The exact procedure in process instance classification is actually partial sequence classification associated to each process instance. SVM classifier predicts one of the next tasks of process instances may face after the branching (in fact, partial sequences) based on the data attached to it. After applying classification on all possible route branching to the end of the process model, the future route for the process instance is made. In the training phase of SVM classifier, the partial trace and the data associated to each completed process instance are given as input to the classifier and then the classifier outputs the predicted next task for the process instance.

The route of each process instance is specified by the partial trace in the event log. Since classifier is trained based on data carried by process instance, therefore, partial trace of each process instance with the last version of data attached The classifier outputs the next predicted task.

By this stage of the algorithm, the route of process instance is predicted. Since the process model is enriched by time information mined from event log, the predicted route is a linear enriched route of process model as it is observed in Figure 5.

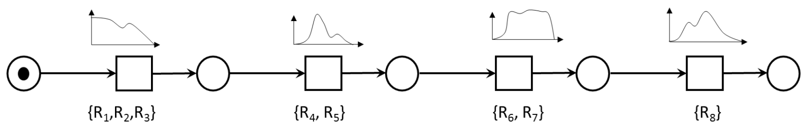

Figure 5: A sample result of a predicted enriched linear route

```
$RoutePredictionAlgorithm:$
$Path \quad PathPredictionAlgorithm(ProcesModel\ P:<>,\ \ Case\ case,\ \ Activity\ A_C)\{$
$partialTrace \ \leftarrow < A_C >$
$while\ (A_C\ ! = P.END)\{$
  $if\ ((A_C,\ A_{next}) \in L)\ and\ (A_{next} \in A))\{$
    $partialTrace.push\,(A_{next})$
    $A_C \leftarrow A_{next}$
  $\}elseif\,(\ (A_C, C_{next}) \in L)\ \ and\ (C_{next} \in C))\{$
    $SVMClassifier \leftarrow \partial\,(C_{next})$
    $A_{next} \leftarrow\ \ SVMClassifier\,(case,\ A_C)$
    $partialTrace.push\,(A_{next})$
    $A_C \leftarrow A_{next}$
$\}$
```

4.4. Fourth Phase: Iterative Synchronization Algorithm

To resolve the issue of waiting time increase and cycle time increase in each pair of process instances with hidden processing dependency, a synchronization algorithm is presented based on the presented process state prediction algorithm to make each pair of dependent process instances arrive at the corresponding pair of tasks simultaneous or near-simultaneous. Therefore, the aim of synchronization algorithm is to minimize the difference time between reaching each pair of dependent tasks. To this end, the synchronization algorithm requires iteratively re-estimating the remaining time to the synchronization target points for each of the two process instances. The synchronization algorithm is iteratively done at each step wherein the process instance takes a step forward in the process model. The approach is to minimize the difference between the time predictions of the two process instances to a pair of corresponding tasks called as “Synchronization delay”. Minimization of synchronization delay includes rearranging the work items in the work list in order to prioritize the posterior work item in the work list so that the work item is set a number of work items earlier.

The new position of posterior work item in the work list is computed fairly. If there exist more than one work list ahead to the synchronization point for the work item, it would definitely not fair to compensate the whole synchronization delay in the first single work list, the delay is distributed fairly between the work lists ahead. The remaining time is comprised of processing times of tasks to the synchronization point and waiting times in the work lists to the synchronization point. First, the sum of processing tasks ahead is subtracted from predicted remaining time of posterior work item; then, the result is divided to sum of waiting times of all tasks ahead (notice waiting time of each task is computed by multiplication of processing time of the task by the number of work items); The result of division represents the fair ratio. By multiplying the fair ratio by waiting time of each work list, the position for posterior work item in the work list is obtained.

$$FairRatio = \frac{PredictedRemainingTime - \sum_{for\ all\ tasks\ ahead} ProcessTime}{\sum_{for\ all\ tasks\ ahead} (ProcessTime \times Num(workitems))}$$

$$PosteriorNewPosition = \ Round(FairRatio\ \times (ProcessTime \times Num\,(workitems)))$$

Since the synchronization algorithm is an iterative algorithm, the above procedure for the new position of the posterior work item is iterated in every task ahead.

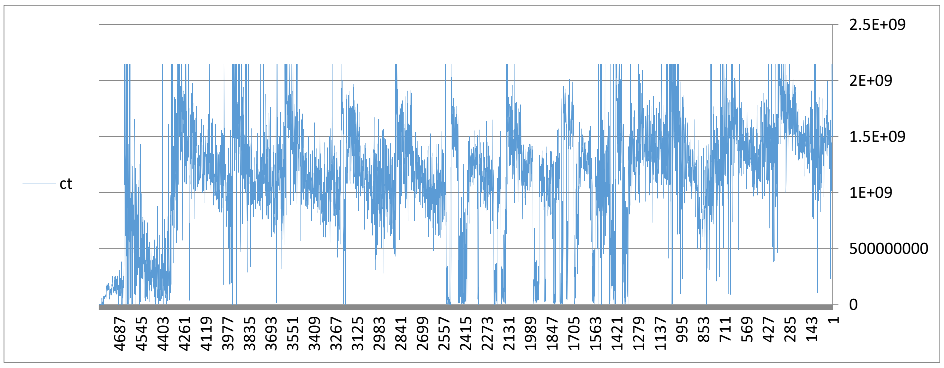

Figure 6: Cycle time of business processes of BPI challenge 2012 event log.

5. Experimental Results

In Figure 6, cycle time of 4720 business processes of BPI challenge 2012 real-life event log is plotted. As it can be observed, the progress of business process cycle times is not smooth in ascending and descending curves and it experiences hard noise and "outlier" like oscillations. On the other hand, a special kind of dependency between business processes is introduced and discovered which leads to cycle time increase in corresponding business processes. By synchronization of business processes, it would result in a more normalized and balanced plot, but it should be noted that not all hard noises in cycle time plot is related to dependent business processes.

In the Figure 7, the ratio of waiting time of dependent business processes to overall cycle time of all business processes for different arrival rates is computed in normal condition without taking dependent business processes into account. The normal base resource allocation is compared with the proposed approach of normalization of cycle times by synchronization of dependent business processes with two different implementations; In one implementation, the corresponding business processes are considered as the 1^{st} ranked priority in each work list and in the other implementation, the corresponding business processes are randomly ranked from 1^{st} to 3^{rd} in each work lists. The two implementation strategy by different priority rankings is originated from the fact that priority of synchronization procedure could not be set as 1^{st} in any condition and higher priorities exist. The figure is plotted for different inter-arrival rates of 3 to 15. As it is observed, as inter-arrival rate decreases, the waiting time of dependent business processes increases. This is due to the fact that anomalies and congestions emerge by decreasing inter-arrival rate and consequently increasing number of business processes.

Since the synchronization procedure is an action for which earliness is of important factor, the number of tasks between starting point of synchronization procedure to the synchronization target point -which is called synchronization path- effects on the synchronization delay result. In Figure 8, the mean absolute error between the predefined target arrived time and actual arrived time is computed for the corresponding dependent business processes in the event log for different lengths of synchronization path. As it is observed, when two tasks are in the synchronization path, the MAE error is high because it is too late to synchronize the waiting time of dependent business processes. As the synchronization path increases to 5 tasks, the minimum MAE error is achieved and the best result is obtained. Finally, as the synchronization path increases to 7 tasks, the minimum-delay synchronization is again hard to achieve because the synchronization is a prediction-based procedure and the underlying conditions change over long time.

In Figure 9, the effect of the synchronization on the overall cycle time is computed for different

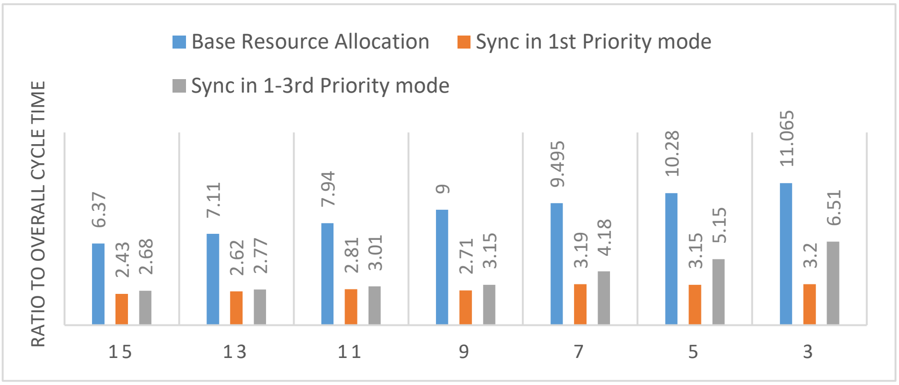

Figure 7: The ratio of waiting time of dependent business processes to overall cycle time for all business processes for different arrival rates

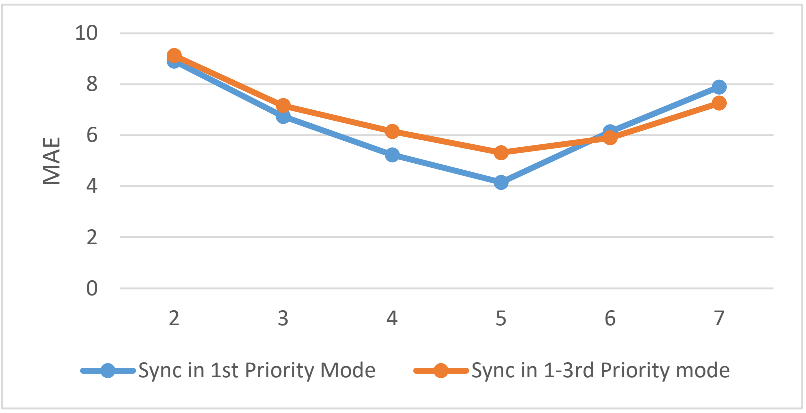

Figure 8: Prediction accuracy (measured in terms of mean absolute error[MAE]) across different lengths of synchronization path

inter-arrival rates. The two implementations for hard priority strategy and soft priority strategy are both evaluated against the overall cycle time and it is shown that hard priority strategy outperforms soft priority strategy in higher inter-arrival rates.

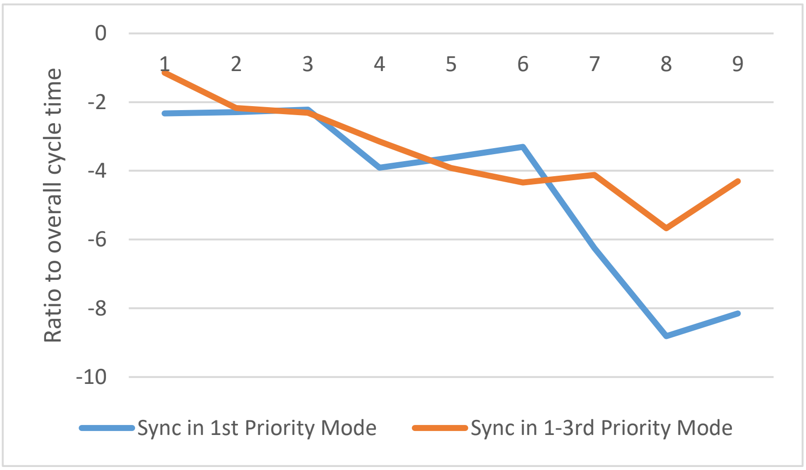

Figure 9: The cycle time reduction percentage rate of the proposed approach for two different priorities

6. Conclusion

In this paper, a new kind of dependency between business processes is discovered and defined for which synchronization procedure is necessary. By Applying the synchronization procedure, not only cycle time reduction is achieved but also cycle time of business processes are normalized. The dependency is defined as a condition in which a process instance waits for the corresponding process instance to trigger at the corresponding task. Synchronization Procedure is a 4-phase approach and in the first phase, the process model is mined from the event log and enriched by the probabilistic distributions of time information. In the second phase, the hidden processing dependency between the each pair of dependent process instances is formally defined and is mined from the event log. In the third phase, a process state prediction algorithm is presented to predict the future route of process instance and then predict the remaining time of the process instance to a given point in a predicted route of the business process. In the fourth phase, an iterative synchronization algorithm is presented base on the presented process state prediction algorithm to make each pair of dependent process instances arrive at the corresponding pair of tasks simultaneous or near-simultaneous. Experimental results on a real-life event log of BPI challenge 2012 show that the proposed method leads to 4.3% reduction in overall cycle time and 39% reduction in cycle time, specifically for dependent process instances.

.

7. References

1. Hammer, M., The agenda: What every business must do to dominate the decade. 2003: Crown Pub.
2. Hammer, M. and J. Champy, Reengineering the Corporation: Manifesto for Business Revolution, A. 2009: Zondervan.
3. Smith, H. and P. Fingar, Business process management: the third wave. Vol. 1. 2003: Meghan-Kiffer Press Tampa.
4. Van Der Aalst, W.M., A.H. Ter Hofstede, and M. Weske. Business process management: A survey. in International conference on business process management. 2003. Springer.
5. Xu, J., C. Liu, and X. Zhao. Resource allocation vs. business process improvement: How they impact on each other. in BPM. 2008. Springer.
6. Huang, Z., X. Lu, and H. Duan, A task operation model for resource allocation optimization in business process management. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2012. 42: p. 1256-1270.
7. Huang, Z., et al., Reinforcement learning based resource allocation in business process management. Data & Knowledge Engineering, 2011. 70: p. 127-145.
8. Wang, J. and A. Kumar, A framework for document-driven workflow systems, in Business Process Management. 2005, Springer. p. 285-301.
9. Zhou, Z., S. Bhiri, and M. Hauswirth. Control and data dependencies in business processes based on semantic business activities. in Proceedings of the 10th international conference on information integration and web-based applications & services. 2008. ACM.
10. Sangal, N., et al. Using dependency models to manage complex software architecture. in ACM Sigplan Notices. 2005. ACM.
11. Lee, K. and K.C. Kang. Feature dependency analysis for product line component design. in International Conference on Software Reuse. 2004. Springer.
12. Zhang, W., H. Mei, and H. Zhao. A feature-oriented approach to modeling requirements dependencies. in 13th IEEE International Conference on Requirements Engineering (RE'05). 2005. IEEE.

13. Wu, Q., et al. Categorization and optimization of synchronization dependencies in business processes. in 2007 IEEE 23rd International Conference on Data Engineering. 2007. IEEE.
14. Barga, R.S. and C. Pu. A Practical and Modular Implementation of Extended Transaction Models. in VLDB. 1995.
15. Van Der Aalst, W.M. and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. in International Workshop on Web Services and Formal Methods. 2006. Springer.
16. Orriëns, B., J. Yang, and M.P. Papazoglou. A framework for business rule driven service composition. in International Workshop on Technologies for E-Services. 2003. Springer.
17. Reiss, S.P. Constraining software evolution. in International Conference on Software Maintenance, 2002. Proceedings. 2002. IEEE.
18. Elmagarmid, A.K., Database transaction models for advanced applications. 1992: Morgan Kaufmann Publishers Inc.
19. Jajodia, S. and L. Kerschberg, Advanced transaction models and architectures. 2012: Springer Science & Business Media.
20. Andrews, T., et al., Business process execution language for web services. 2003, version.
21. Van der Aalst, W., T. Weijters, and L. Maruster, Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 2004. 16: p. 1128-1142.
22. Weijters, A., W.M. van Der Aalst, and A.A. De Medeiros, Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP, 2006. 166: p. 1-34.
23. De Medeiros, A.A. and A. Weijters. Genetic process mining. in Applications and Theory of Petri Nets 2005, Volume 3536 of Lecture Notes in Computer Science. 2005. Citeseer.