Extracting Accurate Performance Indicators From Execution Logs Using Process Models

Nesma M. Zaki, Ahmed Awad, and Ehab Ezat

Information System Department, Faculty of Computers and Information,
Cairo University, Egypt
{n.mostafa,a.gaafar,e.ezat}@fci-cu.edu.eg

Abstract. Performance indicators (PIs) are defined to measure the progress towards the achievement of goals. The input to calculate these measures are the daily operations of an organization. Business processes are the means by which an organization operationalizes the achievement of its goals. So, it is logical to measure PIs based on business processes performance. Processes manifest their performance in execution logs where process and context data are recorded. In this paper, we introduce a new approach that uses information stored in process execution logs in order to extract more detailed and accurate performance measures. We show how the inclusion of information about activity lifecycles and process models can help achieve that. In addition, we evaluate our approach against similar approaches by using both real and synthetic logs.

Keywords: Process logs, performance measurement, business processes, log analysis

1 Introduction

Organizations tend to establish their strategic goals in order to keep moving forward, get motivated to do more and maintain success of their businesses [24]. Measuring the level of fulfillment of these goals is difficult as these goals are unquantifiable measures. Thus, organizations define quantifiable measures whose assessment reflects the level of achievement of goals. These measures are called Performance Indicators (PIs) [27,14]. PIs are for top management in the form of aggregated measures about the daily transactional level performance [5]. These could be related to throughput, service time, work loads of employees etc.

On the other hand, to achieve goals, organizations fragment those goals into perceptible objectives, i.e. operational goals. Then business processes are defined or adapted to fulfill those objectives [28,8]. Thus it makes sense to evaluate the organizational performance based on the performance of its running business processes. Each business process that is enacted produces what is called execution log. An execution log carries all details about the activities that have been executed, their timestamp, people participated into the execution and data that have been processed. The quality of the log depends on the level of maturity and automation of process execution within the organization [22].

From logs, we are able to measure PIs [2,14,22]. For example, workload for all resources within time period, synchronization time and cycle time are among performance indicators that can be extracted from an execution log. Also, these PIs can be defined in different dimensions which are time, cost, quality and flexibility dimensions. Several approaches ,e.g. [2,10,25], have investigated the use of process logs in order to extract performance measures. However, these approaches have considered the log as the only source of information.

In this paper, we propose an approach that uses information stored in event logs as well as the knowledge about process model, that produced these logs, for extracting more detailed and more accurate performance measures. We currently address the time dimension of PIs. This approach focuses on three perspectives: resource, case and activity. It also takes into consideration the combination of these perspectives to obtain more advanced metrics that help in the performance analysis. We currently assume that those logs are generated from an automated execution of the processes, i.e. via a central execution engine and thus logs are information-rich and record all instances and the context data. We use process models to learn the control flow relationships among the different activities in order to get accurate performance measures on the level of a case, activity instance.

The rest of this paper is organized as follows: Section 2 overviews our approach and briefly discusses some of the background concepts and techniques that are used throughout the paper. Section 3 presents our contribution in measuring PIs from process execution logs. In Section 4, we evaluate the proposed approach against other approaches. Related work is discussed in Section 5. A critical discussion of the approach and an outlook on future work is presented in Section 6.

2 Overview & Background

In this section we give an overview of the proposed approach. We measure four types of time-based PIs. These are *service time*, *effective time*, *waiting time* and *sojourn time*. These PIs are measured for four dimensions, namely: *process*, case, activity and *resource* and any of their combinations.

The *service time* measure in its general form defines the time elapsed between the point in time when an object has been created in execution environment and the point in time when it has been terminated. An object could be a case, an activity, or a resource. The calculation of service time for cases and activity occurrences is a straightforward. For resources, we do not calculate service times based on the execution log. Rather we expect that as an input to represent the capacity of the different resources in the period covered by the execution log under analysis. The capacity here is seen as the total number of hours a resource was available for work within a certain time period. The *waiting time* measure is defined as the idle time of an activity when it has been offered till it has been started. For resources, the waiting time is the amount of time the resource was not working on any activity and also not having activities in his work list. The *sojourn time* measure is defined as the time elapsed between the point in time when an activity has been offered and the point in time when it has been terminated. This measure applies to the combination of activity and case dimensions only. The

effective time measure excludes from the service time those intervals where the object was idle, waiting time, e.g., suspended or allocated and waiting for starting.

The calculation of the *effective time* of an object is different from the *service time* only if the execution log records more details about state transitions of the activities. The more details the log contains the more accurate effective time measures that can be obtained. We elaborate more on the calculation of the measures in Section 3.

Figure 1 summarizes the way our approach works in the form of a BPMN process. There are three inputs, an execution log, the execution lifecycle model and the process model that has been enacted and for which the log was recorded. The lifecycle model defines the schema against which the events are generated and recorded in the log. For the work presented in this paper, we fix that input to a reference lifecycle model which is a comprehensive lifecycle developed by Russel et al. [23]. We elaborate more on this model in Section 2.1. The need for the knowledge about the lifecycle model is important for the measurement of some of the PIs. The process model is the third input. The knowledge about the process model is essential to calculate case measures. Actually what we need from the process model is the set of activities and the structural relationships among these activities. For this, the abstraction provided by the notion of the refined process structure tree (RPST) [26,20] is an appropriate abstraction that provides this information. The pre-processing step in Figure 1 obtains the RPST and makes it ready for further steps of performance measures calculation.

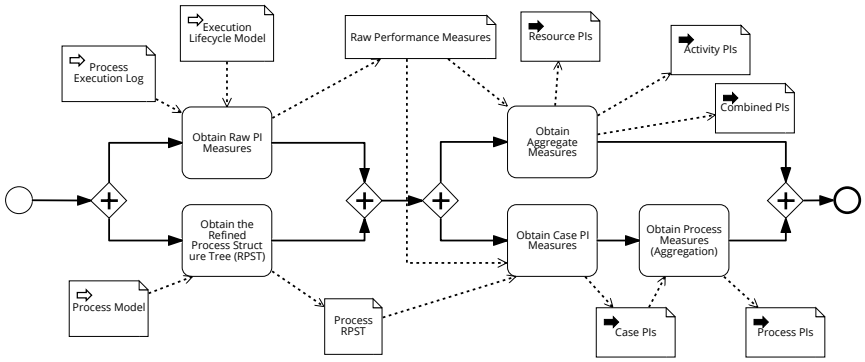

Fig. 1. Overview of the approach

The first processing step is to compute what we call the *raw performance measures*. These measures are calculated for the *effective*, *service* and *waiting* time PIs. For the *sojourn* time PI, it is calculated separately according to pairs of activity and case dimensions only. Based on these raw measures, more aggregated ones like resource, activity, case, process, or other arbitrary combinations can be computed. To help clarify the idea, we establish an analogy between the raw performance measures and fact tables in data marts star schema [11]. In our case, the raw performance measure stores the value of the measure per all combinations of the dimensions values. In our case, the dimensions are, the process, activity, resource, process-instance (case). Definition 1 formalizes the notion of raw performance measures.

Definition 1 (Raw Performance Measure). *Let P be the set of process models deployed for execution, C be the set of all cases that have been recorded in a log, A be the set of all activities, R be the set of all resources that participate in process execution. A Raw Performance Measure (rw) is a tuple on the form:*
$(p, c, a, r, occurrence, type, start, end, value)$ *where:*

- $p \in P$ *is the process model for which this measure is calculated,*
- $c \in C$ *is the case for which this measure is calculated,*
- $a \in A$ *is the activity for which this measure is calculated,*
- $r \in R$ *is the resource that contributed in performing* a*,*
- $occurrrence \in \mathbb{N}$ *defines the iteration number in which activity* a *was executed,*
- $type \in \{E, S, W\}$ *defines the measure that is calculated.* E *for effective ,*S *for service time measure and* W *for waiting time,*
- $start$ *is the timestamp of the case* c *start,*
- end *is the timestamp of the case* c *termination,*
- $value \in \mathbb{R}$ *is the value of the measure*

We define the set RW as the set of all raw performance measures.

The property $occurrence$ in Definition 1 accounts for the fact that a certain activity might be executed an arbitrary number of times either due to loops or redoing the activity, cf. Figure 2. The properties $start$ and end are the timestamps corresponding to the case c start and end respectively. These depend on the case lifecycle where we expect a separate event to signal the creation of a new case and another event to signal the termination of a running case. These properties might be needed if the user wants to extract measures with a specific time window within the coverage of the execution log. The property $type$ defines that basically *three* raw performance measures can be computed. Note that the value of each type is calculated at the finest level of combining case c, activity a and resource r. Note also that the process identifier p is functionally dependent on case c as well as the $start$ and end properties.

To obtain measures for certain dimensions, e.g., resource, activity, or case, we need to apply aggregations over members of the RW. For resources and activity dimensions, the aggregation could be as simple as summing up the $value$ property in the relevant elements in RW. Also, other statistical measures can be obtained easily, e.g., *min, max, avg*. For obtaining case measures, aggregation is not as simple as summation. To help explain that, imagine two tasks A and B that are running in a parallel block within a process. In such scenario, the effective time of A and B contributing to case c effective time is not the sum of the effective time of the individual activity occurrences within the case. Rather, by knowing, based on the process model, that those activities were running in parallel. Therefore, the effective time for them together will be the *maximum* value of their individual effective time within the same case. Therefore, our approach requires as an input the process model definition. We calculate the case effective time once and store it for several lookups afterwards. Definition 2 formalizes the notion of case effective time measure.

Definition 2 (Case Effective Time Performance Measure). *Let P be the set of process models deployed for execution, C be the set of all cases that have been recorded in a log. A Case Effective Time Performance Measure (crw) is a tuple on the form:*
$(p, c, value)$ *where:*

- $p \in P$ *is the process model for which this measure is calculated,*
- $c \in C$ *is the case for which this measure is calculated,*
- $value \in \mathbb{R}$ *is the value of the measure*

We define the set CRW *as the set of all case effective time raw performance measures.*

In the rest of this section we briefly describe background knowledge needed to help understand our approach. In Section 2.1 we describe the reference lifecycle model we adopt. In Section 2.2 we describe RPST and how preprocessing of process models works. In Section 2.3, we describe the running example that will serve throughout the rest of the paper. Section 3 will explain in details our algorithms to compute the raw performance measure as well as the calculation of other aggregated measures.

2.1 Reference Life Cycle

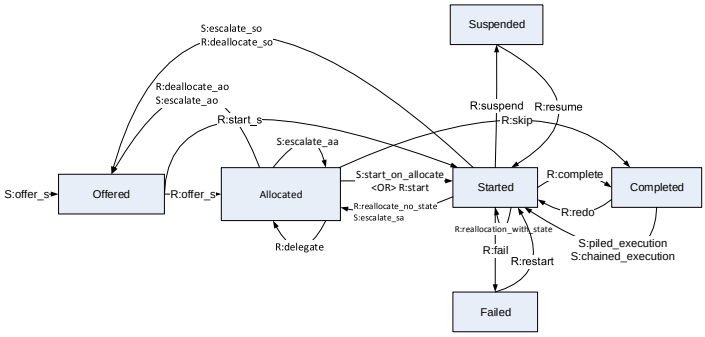

Fig. 2. Task life cycle

In this section, we describe the reference activity life cycle for human activities[1], see Figure 2 adapted from [23]. This life cycle should be interpreted as follows: once an activity is created, the system offers it to one or more resources. At this point the activity is in the *offered* state. Then, one of these resources picks the activity for execution and thus the activity gets *allocated* to that resource. Next, a resource can *start* working on an activity. Then, the resource can temporarily *suspend* the activity. After that he can resume the activity to continue work. Alternatively, the resource can drop the work on the activity. Then, this activity is reallocated to another resource to work on it. Eventually, the working resource can *complete* the activity successfully or *fails* to complete the activity. All these states are represented as entries in the process execution log.

2.2 Refined Process Structure Tree

Calculation of case effective time measures requires the knowledge of the structural relationship among activities within the enacted process. We rely on the concept of a

[1] We refrained from automatic activities as we assume that their impact on the overall case performance is negligible.

refined process structure tree (RPST), which is the process analogue of abstract syntax trees for programs. The concept of a RPST is based on the unique decomposition of a process model into fragments. Fragments, which are the decomposition result, are organized into a hierarchy according to the nesting relation. This hierarchy is called a process structure tree. RPSTs can be constructed using various algorithms. One approach is a decomposition into canonical single entry single exit (SESE) fragments, formally described in [26]. Informally, SESE fragments can be defined as fragments with exactly one incoming and one outgoing edge. The node sets of two canonical SESE fragments are either disjoint, or one contains the other. Following [26], we consider the maximal sequence of nodes to be a canonical SESE fragment. The level of structuring detail that can be obtained from RPST is related to the degree of structuredness of the input process model. If the process contains parts that are not block structured, the resulting RPST will contain so called *rigid* components. The structural relationship between elements within a rigid cannot be determined. However, techniques as in [19,16,18,17,15] may restructure un/semi-structured process models. However, it has not been proven that it will always restructure it. Thus, this remains a limitation. In that case, the measures are summed.

Definition 3 (RPST). *A process structure tree* $RPST = (N, r, E, Type)$ *where:*

- N *is a finite set of nodes, where nodes correspond to canonical SESE fragments,*
- $r \in N$ *is the root of the tree,*
- $E \subseteq (N \times (N \setminus \{r\}))$ *is the set of edges. Let tree nodes* $n1, n2 \in N$ *correspond to SESE fragments* $f1$ *and* $f2$*, respectively. An edge leads from* $n1$ *to* $n2$ *if SESE fragment* $f1$ *is the direct parent of* $f2$*,*
- $type : N \rightarrow \{act, seq, and, xor, or, loop\}$ *is a function assigning a type to each node in* N*: act corresponds to activities, sequences, and, xor, or blocks of corresponding type, loop*

2.3 Running Example

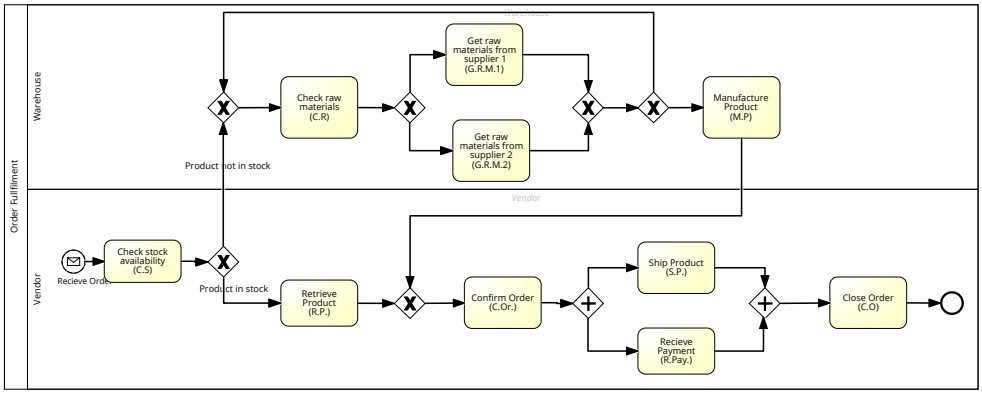

Fig. 3. Order Fulfillment

In this section, we describe an order fulfillment process adapted from [8]. As illustrated in Figure 3, the process starts when a seller receives an order. Then, the seller

Table 1. Sample event log for the order fulfillment process from Figure 3

Event ID	Case	Activity	Resource	Event type	Timestamps
⋮	⋮	⋮	⋮	⋮	⋮
26	1	S.P.	Kareem	Offered	18-03-2015 06:28
27	1	S.P.	Kareem	Allocated	18-03-2015 07:28
28	1	S.P.	Kareem	Started	18-03-2015 09:28
29	1	S.P.	Kareem	Suspended	18-03-2015 15:28
30	1	S.P.	Kareem	Started	18-03-2015 17:28
31	1	S.P.	Galal	Offered	18-03-2015 18:28
32	1	S.P.	Galal	Allocated	18-03-2015 19:28
33	1	S.P.	Galal	Started	19-03-2015 10:28
34	1	S.P.	Galal	Completed	19-03-2015 15:28
⋮	⋮	⋮	⋮	⋮	⋮
54	2	G.R.M.2	Ramy	Offered	14-03-2015 11:46
55	2	G.R.M.2	Ramy	Allocated	14-03-2015 12:46
56	2	G.R.M.2	Ramy	Started	14-03-2015 14:46
57	2	G.R.M.2	Ramy	Suspended	14-03-2015 17:46
58	2	G.R.M.2	Ramy	Started	14-03-2015 18:46
59	2	G.R.M.2	Marwan	Allocated	15-03-2015 08:46
60	2	G.R.M.2	Marwan	Started	15-03-2015 09:46
61	2	G.R.M.2	Marwan	Completed	15-03-2015 11:46
⋮	⋮	⋮	⋮	⋮	⋮
593	14	G.R.M.2	Marwan	Offered	08-04-2015 02:41
594	14	G.R.M.2	Marwan	Allocated	08-04-2015 03:41
595	14	G.R.M.2	Marwan	Started	08-04-2015 05:41
596	14	G.R.M.2	Marwan	Suspended	08-04-2015 11:41
597	14	G.R.M.2	Marwan	Started	08-04-2015 13:41
598	14	G.R.M.2	Marwan	Completed	09-04-2015 02:41
⋮	⋮	⋮	⋮	⋮	⋮

checks the availability of the product in the stock. If the product is in stock, it will be retrieved from warehouse. Then, the order is confirmed. Thereafter, the product is sent to the customer and seller receives the payment before closing the order. If the product is not in stock, seller needs to check the availability of the raw materials at suppliers in order to manufacture the product. Depending on the product, raw materials will be ordered either from *supplier 1* or *supplier 2*. If any of raw materials are not available, then seller will check again the availability of these materials from another supplier. Once all materials are available, the product will be manufactured. Then,the process continues with confirming the order and the rest of the steps.

Table 1 is part of an execution log that is generated from the *Order Fulfillment* process in Figure 3. This log contains all details about activities that have been executed, the state transitions of the activities, resources participated into execution and timestamps, etc.

3 Calculating Performance Measures

In this section, the four measures of time-based PIs are defined which are effective time, service time , waiting time and sojourn time with the different dimensions discussed before, resource, activity, case, process and their combinations.

3.1 Effective Time

Effective time can be computed as the consumed time between *started* and *completed* events of activity after excluding suspension time based on the activity life cycle in Figure 2. Based on Definition 1, the effective time measure is calculated per the combination of *case*, *activity*, *resource* and activity *occurrence*. This gives the finest level of details that can be later on aggregated to deliver a value from the point of view of one dimension or more. The combination of an activity and a resource is necessary because it might be the case that more than one resource can work on the same activity instance. Thus, we are able to distribute the shares of a single activity occurrence between the different contributing resources.

Algorithm 3.1 illustrates how to calculate Effective time raw performance measure.

Algorithm 3.1 Calculating effective time raw performance measure

Input: A process Model identifier p
Input: Execution log L for a specific process model p
Output: RW The effective time raw performance measure
$T_s = null$ {start of work timestamp}
$T_e = null$ {end of work timestamp}
$occurrence = 0$
$C = CASE(L)$ {get the list of all unique case identifiers within L}
1: **for** $c \in C$ **do**
2: $start = get\_start\_time(c)$; $end = get\_end\_time(c)$
3: $A = ACT(c)$ {get the list of all unique activity identifiers within case c}
4: **for** $a \in A$ **do**
5: occurrence = 1; r = null; {r represents current resource}
6: $X \subset L = \{l \in L | l.case = c \wedge l.activity = a\}$
7: **for** $j \in X$ **do**
8: **if** (r is null) **then**
9: r = j.resource;
10: **if** ($j.resource == r$) **then**
11: **if** ($j.event\_type == started$) **then**
12: $T_s = j.timestamp$;
13: **if** ($X(j).event\_type \in \{completed, failed\}$) **then**
14: $T_e = j.timestamp$
15: $Eff\_time = Eff\_time + (T_e - T_s)$
16: $RW = RW \cup \{(p, c, a, r, occurrence,' E', start, end, Eff\_time)\}$
17: occurrence = occurrence + 1;
{In case of resuming the activity}
18: **if** ($j.event\_type == suspended$) **then**
19: $T_{susp} = j.timestamp$
20: $Eff\_time = Eff\_time + (T_{susp} - T_s)$
21: **else if** ($j.resource \neq r$) **then**
22: **if** ($T_s \neq null$) **then**
23: **if** ($j.event\_type \in \{offered, allocated, started\}$) **then**
24: $T_e = j.timestamp$
25: $Eff\_time = Eff\_time + (T_e - T_s)$
26: $RW = RW \cup \{(p, c, a, r, occurrence,' E', start, end, Eff\_time)\}$
27: r = j.resource;
28: **if** (T_s is null) **then**
29: r = j.resource;

To explain how the algorithm works, assume that the input log L to the algorithm is Table 1 and the process P is the one shown in Figure 3. First, we define the set C to hold the identifiers of the different cases in the log. In our case, $C = \{1, 2\}$. By iterating on each case $c \in C$, we obtain the start and end time of the case. These correspond to the timestamp of the event signaling the start of the case and the event signaling its termination respectively, cf. Line 2. Note that the end time can be $null$ if the case was not completed by the time the log was put into analysis. Next, we obtain the set of

activity identifiers A that have been executed within case c. In our example for $c = 1$, $A = \{$C.S., R.P., C.Or., S.P., R.Pay, C.O.$\}$. We now iterate over each activity $a \in A$ obtaining the set X ,in Line 6. This set is a subset of the log L where only entries, events, related to the combination of c and a are retained. Iterating over $j \in X$, we start calculating the effective time measure. X is sorted chronologically by the timestamp.

Effective time is calculated by summing the time intervals in which activity a was being processed by resource r within case c. These intervals are the time elapsed between a) resource r *started* working on a and the time he has *completed* the task, or b) *start* time of work of r on a and the time at which r *suspended* his work on a, or c) r *started* working on a and the time when r *failed* to complete a, or d) r *started* to work on a and the time r decided to give up working on a, Line 21, either by *offering*, *allocating* or *starting*, Line 23, a to some other resource. At the times of *completion or failure*, the effective time raw performance measure is updated, cf. Definition 1, Line 16. Also, if the processed log was taken while an activity was still under processing, i.e., neither a *completed* nor a *failed* event was logged for it, the so-far computed *effective time* is added to the raw performance measure, Line 26.

Algorithm 3.1 also takes loops into consideration by checking the states of activity being either completed or failed. Each time an activity was completed or failed within the same case, the $occurrence$ is incremented as shown in Line 17. Note that recurrence of an activity might be due to loops in the process definition or due to restart of the same activity instance as permitted by the activity lifecycle, cf. Figure 2.

Table 2. Raw performance measures for the log from Table 1

#	P.ID	C.ID	Activity	Resource	Occurrence	type	start	end	value
1	1	1	S.P.	Galal	1	Effective	07-03-2015 6:28	22-03-2015 9:28	5
2	1	1	S.P.	Kareem	1	Effective	07-03-2015 6:28	22-03-2015 9:28	7
3	1	1	S.P.	Kareem	1	Service	07-03-2015 6:28	22-03-2015 9:28	12
4	1	2	G.R.M.2	Ramy	1	waiting	07-03-2015 11:46	30-03-2015 11:46	3
5	1	2	G.R.M.2	Marwan	1	waiting	07-03-2015 11:46	30-03-2015 11:46	1
6	1	14	G.R.M.2	Marwan	1	service	24-03-2015 2:41	02-05-2015 2:41	23
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

To give an example, we calculate effective time for *Kareem* according to activity *S.P* in case 1. As shown in Table 1, *Kareem* started this activity at time $t_s = 18-03-201509:28$ with $EventID$ 28. He suspended this activity at time $t_e = 18-03-2015\ 15:28$ with $EventID$ 29. After that, he resumed the activity at time $t_{s'} = 18-03-2015\ 17:28$ with $EventID$ 30. For some reason, he did not complete this activity and it was reoffered to another resource *Galal* at time $t_{e'} = 18-03-2015\ 18:28$ with $EventID$ 31. Therefore effective time that *Kareem* performed this activity was 7 hours (t_e - t_s)+ ($t_{e'}$ - $t_{s'}$).

After storing all information in raw performance measures as seen in Table 2 [2], we can compute more aggregated measures with respect to the resource, case, activity or other combinations. In order to obtain these measures, we need to apply aggregation functions on the raw performance measures.

[2] E stands for Effective time measure whereas S stands for Service time and W stands for Waiting time in Table 2

Definition 4 (Performance Measure Selection). *Let A be the set of all activities, let R be the set of all resources, let C be the set of all cases, let P be the set of all Processes, let RW be the set of all raw performance measures, let $SE \subset (A \cup \{\perp\}) \times (R \cup \{\perp\}) \times (C \cup \{\perp\}) \times (P \cup \{\perp\}) \times \{E, S, W\}$ be the set of filter selectors over RW, we define the performance measure selection $\sigma_{(a,r,c,p,t)\in SE}(RW) = \{rw \in RW | (a = rw.activity \vee a = \perp) \wedge (r = rw.resource \vee r = \perp) \wedge (c = rw.case \vee c = \perp) \wedge (p = rw.process \vee p = \perp) \wedge rw.type = t\}$*

Definition 4 formalizes how a selection over the set of raw performance measures can be made by filtering with the activity, resource, case, and the type of measure being effective time, service time or waiting time. The value $\perp$ indicates that the selected dimension has not been filtered for. A selection on the form $(\perp, \perp, \perp, \perp, E)$ means that we retrieve all tuples of the effective time measure for all activities, resources, cases , process in scope. Whereas, a selection on the form $(\perp, r1, \perp, \perp, E)$ means that we select all tuples for resource $r1$ only.

Definition 5 (Aggregate Performance Measure). *Given that $\Omega = \sigma_{(a,r,c,p,t)\in SE}(RW)$ is the set of selected elements from RW, we define the aggregate performance measure $AGG(\Omega) = G_{rw\in\Omega}(rw.value)$.*

Definition 5 formalizes the notion of aggregating performance measure based on a selection result over RW. Note that G is a placeholder for an aggregation function that could be different based on the chosen selectors. For instance, for selectors over resource only, aggregation would be by iterating over Ω and summing the *value* property. Similarly, are aggregations over activity selectors . Aggregation over the *case* selector only is more sophisticated than simply summing the value over the case entries.

Case Effective Time Calculating Effective time for *case* selectors depends on the raw performance measures (RW) and also on the RPST representation of the input process model P. As RPST helps determine parallel branches, we can get accurate effective time results per case by getting the *maximum* values among the different parallel branches. Algorithm 3.2 illustrates how to compute effective time for case using RPST.

Algorithm 3.2 Calculate case effective time aggregate measure

Input: RW
Input: RPST
Input: P
Output: Effective time for case (CRW)
$C = CASE(RW)$ {get the list of all unique case identifiers within RW}
1: **for** $c \in C$ **do**
2: $A = ACT(c)$ {get the list of all unique activity identifiers within case c}
3: **for** $a \in A$ **do**
4: $\Omega = \sigma_{a,\perp,c,E}(RW)$
5: $activityNode = RPST.getNode(a)$ {get activity node from RPST}
6: $activityNode.measures = \Omega.value$ {assign value from selector on activity node in RPST}
7: $CaseEffectiveTime = 0$
{check type of current node r}
8: **if** $type(RPST.r) = and$ **then**
9: $CaseEffectiveTime+ = getMaxEffectiveTime(RPST.r)$
10: **else if** $type(RPST.r) \in \{seq, xor, or, loop\}$ **then**
11: $CaseEffectiveTime+ = getSumEffectiveTime(RPST.r)$
12: $CRW = CRW \cup (p, c, CaseEffectiveTime)$

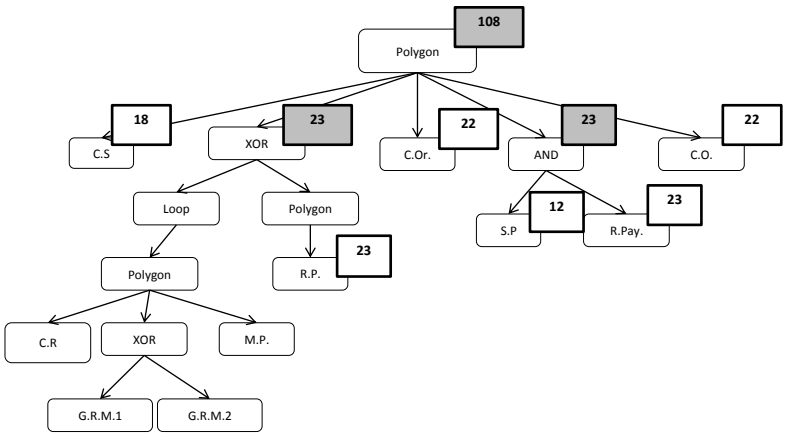

Fig. 4. RPST of Figure 3 instantiated for case 1

Algorithm 3.2 starts by retrieving all cases within the raw performance measure RW. Next, the set of activities within each case is obtained. By iterating over activities within cases, we can apply selectors on RW based on the case c and activity a under investigation, Line 4. Once the set $\Omega \subset RW$ is obtained for that selection, we can populate the respective activity node in the RPST with the values of the effective time measure from Ω. For instance, the effective time for activity *Check Stock Availability* (C.S) within case 1 is 18 hours, cf. Figure 4. Next, the algorithm examines the type of root node r of $RPST$. In case of node type and, the function $getMaxEffectiveTime$ is invoked over children of r. If the node of is other types than and and act, cf. Definition 3, the function $getSumEffectiveTime$ is invoked instead. These two functions are recursive by nature that traverse the tree from the input node downwards. In the first case, the and block case, the effective time is the maximum value among the effective measure of the block's children nodes. In the second case, other block types, the effective measure for the block is the summation of the effective values of the children nodes. Note that for the xor block only one branch can be executed at runtime within a specific cycle, in case the xor block is nested within a loop. At the end, the case effective time measure CRW, cf. Definition 2, is updated with the value. Algorithm 3.2 serves as an implementation of the abstract aggregate function G from Definition 5 to obtain effective time aggregate measure for a case selector.

Figure 4 represents the RPST that contains the effective time of each activity of case 1. We can obtain effective time for case 1 as $18 + 23 + 22 + 22 + Max(23, 12)$ which is equal to 108 hours.

3.2 Service Time

Service time can be defined as the consumed time between observing the event *allocated* or *started* then either observing the event *completed* or *failed* thereafter, cf. Figure 2. This can be easily calculated per case and per activity and stored in the raw performance measure RW. As we mentioned earlier, we treat the resource service time differently,

cf. Section 2, where we require that value as input to our approach. However, we can calculate the service time per the combination of case, activity, and resource based on the input log.

We illustrate by an example how to calculate service time raw performance measure for combination of resource, case and activity. For example, in table 1, we find that *Marwan* allocated activity *G.R.M.2* for case 14 at time $t_s = 08-04-201503:41$ with $EventID$ 594. He completed this activity at time $t_{comp} = 09-04-2015\ 02:41$ with $EventID$ 598. Therefore, service time that *Marwan* took in this activity in case 14 is 23 hours which is the difference between t_{comp} and t_s whereas as the effective time for *Marwan* working on activity *G.R.M.2* in case 14 is just 19 hours. This data also stored in Table 2, see row 6.

Case Service Time is defined as the time elapsed from the start of the case and its end. According to Table 2, we can obtain start and end time for each case. For example, start time, t_{start}, for case 1 is $07-03-2015\ 06:28$ and end time, t_{end}, is $22-03-2015\ 09:28$. Therefore, service time for case 1 is $t_{end} - t_{start}$ which is 363 hours. Note that service time is defined for $completed$ cases only within the log. In other words, if a case i was still running by the time a snapshot of the log was put into analysis, the service time for i is undefined. However, we can obtain service time on a finer granularity as we just discussed in the last paragraph.

Activity Service Time The calculation of *activity* service time can be done by first applying an activity selection $\Omega = \sigma_{(\perp,a,\perp,\perp,S)}(RW)$ and then looking for the very first *allocated* or *started* event timestamp t_s and the very last *completed* or *failed* event timestamp t_e for a activity per each case. Finally, the activity service time measure is the summation of the differences between t_e and t_s. Note that also it is straight forward to obtain more statistical measures like the average service time of the activity. Similarly to the case situation, if the activity completion was not recorded by the analysis time of the log, activity service time is undefined.

3.3 Sojourn time

Sojourn time is defined as the elapsed time between the very first *offered* event and the very last *completed* event. This measure can be computed for the combination of an activity and a case. For example, in Table 1, we find that activity *S.P.* within case 1 was *offered* on $18-03-2015\ 06:28$ with $EventID$ 26. Then, it was *completed* at $19-03-2015\ 15:28$ with $EventID$ 34. So, sojourn time for *S.P* in case 1 is 33 hours.

3.4 Waiting time

Waiting time is defined as the differences in time between every *offered* or *allocated* event and the following *started* event. This can be calculated per case, activity and resource combination at the finest level.

We illustrate by an example how to calculate the waiting time raw performance measure. In Table 1, we find that activity *G.R.M.2* in case 2 was *offered* to *Ramy* at time (t_{offer}) $14-03-2015\ 11:46$ with $EventID$ 54. Then, it was *started* at time (t_s) $14-03-2015\ 14:46$ with $EventID$ 56. Therefore, the waiting time of *Ramy* is 3 hours which is the difference between t_s and t_{offer}. Also, we notice that this activity was reallocated again to *Marwan* at time $15-03-2015\ 08:46$ with $EventID$ 59. Marwan *started* the activity again at time $15-03-2015\ 09:46$ with $EventID$ 60. Therefore, waiting time of *Marwan* within this activity is 1 hour which is difference between $EventID$ 60 and 59. The respective waiting time measures are Table 2 in rows 4 and 5.

4 Evaluation

We have implemented the algorithms [3] presented in Section 3 using Java and relational databases. Also, we have used the jBPT library [21] to parse, restructure and obtain the RPSTs of the process model. We have introduced some modifications to the RPST component to fit with our needs such as identifying loops and also classifying bonds by block type, i.e, XOR and AND.

We found basically three approaches that analyze process performance which are the Performance Analysis with Petri net[9][4], Basic Performance Analysis[5] and event gap analysis [25]. All are implemented as ProM [3] plugins. We found difficulties in running the event gap analysis plugin. Thus, we could not evaluate our approach against it. Our evaluation is based on synthesized and real life logs.

Performance Analysis with Petri Net (PAP) The main objective of this approach is to extract performance information from event logs and process model, Petri nets. This approach replays the log on the Petri net in order to obtain performance information such as sojourn time, execution time , effective time in our terms , waiting time of activity and case combined measures and also, throughput time, service time in our terms, of the case dimension. Also, some statistical values are calculated such as minimum, maximum and average time for each activity and case.

Basic Performance Analysis (BPA) This approach extracts execution time, service time in our terms, and waiting time of the combinations of activity/case, activity/resource, resource/case and for the activity dimension. Also, it extracts the sojourn,service time in our terms, waiting time of the case dimension. This ProM plugin also provides some statistical values like the minimum, average, and maximum execution and waiting time for each activity and case all over the log .

Table 4 compares and clarifies the features of our approach against the other approaches.

[3] https://github.com/NesmaZaki/PIExtraction-

[4] https://tinyurl.com/PerformanceAnalysisWithPN

[5] http://www.processmining.org/online/basicperformanceanalysis

Measures	PAP			BPA			Our Approach		
	Resource	Activity	Case	Resource	Activity	Case	Resource	Activity	Case
Effective time	−	+	−	−	−	+	+	+	+
Waiting time	−	+	−	+	+	+	+	+	+
Service Time	−	−	+	+	+	+	+	+	+
Sojourn Time	−	+	−	−	−	−	−	+	−

Table 4. Feature comparison between three approaches

4.1 Synthesized Log

Synthesized log was generated from the ProM plugin "Perform a simple simulation of (stochastic) Petri net" and we made some modifications to add information resources as well as more events to reflect transition in activities lifecycles.

We compare PAP plugin to our approach using the complete log from Table 1. We generated the Petri net underlying the order fulfillment model based on transformation rules in [6] as it was required by the plugin.

Indicator	Activity/Case	PAP	BPA	Our approach
Execution (Effective) time	S.P./1	5 hours	–	12 hours
Waiting time	S.P./1	17 hours	17 hours	19 hours
Service time	S.P./1	–	30 hours	32 hours
Sojourn time	S.P./1	20 hours	–	33 hours

Table 5. Summary of some results related to activity/case dimensions

Table 5 summarizes aggregated measures for the activity within case dimensions. For example, the aggregated measure that applied for the activity ship product (S.P) is $AGG(\sigma_{(S.P,\perp,1,\perp,t\in\{E,S,W\}}(RW))$ within case 1. We chose that activity measure as this is the one supported by the plugin and our purpose is to compare our values to the plugin values as reported in Table 5. With respect to the execution time indicator, PAP computes this indicator as the difference between the time stamps of the *very last* completed and the *very last* started events for an activity occurrence within a process instance. In our terms, execution time indicator is the effective time. Our approach has accounted for all relative events as per Algorithm 3.1 and then running the aggregation for the activity and case combination. For the specific case of the S.P activity within case 1, PAP shows the effective time as $5\ hours$ whereas our approach goes to $12\ hours$. The reason is that PAP does not account for multiple (re)starts of the same activity instance within the same process instance, which is the case for some of the occurrences of S.P. Referring to the log in Table 1, we notice that activity S.P. was first started at time 18-03-2015 09:28 with $EventID$ 28. Next, it was suspended and restarted again with $EventID$ 30. Also, at $EventID$ 33 there was another restart for the activity but with a different resource before it completes at $EventID$ 34 with timestamp 19-03-2015 15:28.

For the waiting time measure, PAP computes it as summation of the differences between the time stamps of *allocated* and *started* events for an activity within a case.

Our approach computes it as summation of the differences between the time stamps of *offered* and *started* events for an activity within case. For the S.P activity within case 1, PAP shows waiting time as 17 hours whereas our approach goes to 19 hours. The difference between these values is due to that our approach takes into consideration the time between the *offered* event at time 18-03-2015 06:28 with $EventID$ 26 and *allocated* events at time 18-03-2015 07:28 with $EventID$ 27. Also, the time between *offered* at time 18-03-2015 18:28 with $EventID$ 31 and *allocated* at time 18-03-2015 19:28 with $EventID$ 32.

For the sojourn time measure, the difference between values is due to that PAP considers only the difference between the time stamps of *very last* allocated and *very last* completed events as exemplified for S.P activity within case 1. This activity was allocated with $EventID$ 32 at time 18-03-2015 19:28 and was completed with $EventID$ 34 at time 19-03-2015 15:28. In our approach, sojourn time is computed as difference between the very first offered event and the very last completion event of the same activity occurrence within a case. For the example of S.P activity, this would be $EventID$ 26 with time 18-03-2015 06:28 and $EventID$ 34 with time 19-03-2015 15:28. Finally, our approach can compute the service time measure which is not supported by PAP. PAP does not provide measures about the resource perspective and thus is not able to provide measures for combination of performance dimensions, whereas our approach is designed to provide such measures.

We compare BPA plugin to our approach using event log in Table 1. Table 5 shows the comparison results for the activity/case combined measure. With respect to waiting time measure, BPA computed it as PAP plugin. BPA does not compute sojourn time of activity within a case. For service time measure, the differences between the values is due to that BPA considers only difference between the timestamps of *started* and *completed* events. For S.P activity within case 1, it *started* with $EventID$ 28 at time 18-03-2015 09:28 and *completed* with $EventID$ 34 at time 19-03-2015 15:28. Our approach computes it as difference between the timestamps of *allocated* and *completed* events.

Indicator	Resource/Activity	BPA	Our approach
Execution (Service) time	Marwan/G.R.M.2	42 hours	26 hours
Waiting time	Marwan/G.R.M.2	3 hours	4 hours

Table 6. Results of the combined resource/activity dimensions

Table 6 summarizes results of aggregated service and waiting time of the combined resource/activity dimensions. For example, the aggregate measure that applied for resource *Marwan* with activity get raw materials from supplier 2 (*G.R.M.2*) is $AGG(\sigma_{(G.R.M.2,Marwan,\perp,\perp,S)}(RW))$. We can notice the difference in values between these approaches. The reason is that BPA computes the execution time of a resource on a specific activity as the difference between the times of the *very first* started event and the *very last* completed event for this activity within a case and then summing over all cases. BPA does not account for cases where the task might have got reallocated between the very first start and the very last complete events. According to Table 1, we notice that activity *G.R.M.2* was started with resource *Ramy* at time 14-

03-2015 14:46 with $EventID$ 56 and was completed with another resource *Marwan* at time 15-03-2015 11:46 with $EventID$ 61.

For the waiting time measure, the difference between values is due to the fact that BPA did not take into consideration that task might have got reallocated again in between first *started* event and last *completed* event. As seen in Table 1, BPA did not calculate the waiting time of *Marwan* on activity *G.R.M.2* in case 2 which is the time between *allocated* event at time 15-03-2015 08:46 with $EventID$ 59 and the *started* event at time 15-03-2015 09:46 with $EventID$ 60, where we account for it.

Indicator	Resource/Case	BPA	Our approach
Execution (Service) time	Galal/1	30 hours	20 hours
Waiting time	Galal/1	15 hour	16 hour

Table 7. Results of the aggregated measures for resource/case dimensions

Table 7 summarizes results of the aggregated service and waiting time of combined resource/case dimensions. BPA computes the execution time of a resource within a case as the summation of the differences in time between the very first *started* and the last *completed* events of each activity, regardless of which resource executed them. The execution time of resource *Galal* within case 1, in activity *S.P*, is the difference between very *first* started event at time 18-03-2015 09:28 with $EventID$ 28 and the very *last* completed event at time 19-03-2015 15:28 with $EventID$ 34. In our approach, the execution time of *Galal* in this case is the difference between *allocated* event at time 18-03-2015 19:28 with $EventID$ 32 and *completed* event at time 19-03-2015 15:28 with $EventID$ 34. This value is more accurate than BPA approach as we only took when this resource actually allocated. For waiting time measure, the difference between values is due to that BPA computes it as difference between *allocated* with $EventID$ 32 at time 18-03-2015 19:28 and *started* with $EventID$ 33 at time 19-03-2015 10:28. Our approach considers also *offered* event with $EventID$ 31 at time 18-03-2015 18:28.

This plugin also calculates the sojourn time, waiting time and execution time of the individual case dimension. With respect to the sojourn time measure, this plugin computes it as the service time of case in our approach. For execution time of case, this plugin computed it as summation of services times of each activity. While in our approach, we compute it as the summation of the effective time of each activity taking into accounts the parallelism of activities as shown in section 3.1. For example, the execution time of case 1 according to this plugin is 149 hours while in our approach it is 108 hours only. The main advantage of this plugin over our approach is that it calculates execution time based on working hours only.

4.2 Real Logs

We chose two real logs from the BPI Challenges in order to evaluate our approach against other approaches. One of these logs is taken from a Dutch Financial institute

[6]. This log contains data that represented an application process for a personal loan. It contains 13087 cases and with a total of 26200 events. This log exposes three transitions types, *Schedule, Start* and *Complete*. Some tasks have only event *Complete* logged with no start which indicates that there are missing events in the log. Each event contains case id, resource who performed the activity, activity, transition type and time stamp. However, some events in this log have the resource information missing. Therefore, we assume that these events assigned to a *Dummy* resource. We refer to this log as the *Financial* log.

We also use another real life log that is taken from Volvo IT Belgium to evaluate the three approaches. This log contains data that represented the IT incident management system [7]. It contains 7554 traces with a total of 65533 events. It exposes the following transitions types, *In call, Cancelled, Unmatched, Wait, In progress, Awaiting assignment, Assigned, Resolved* and *Closed*. We mapped these transition types to match our lifecycle. They are mapped as follows: In call and In progress as *start*, Wait as *suspend*, closed and resolved as *complete*, awaiting assignment as *offer*, assigned as *allocate* and finally, cancelled as *fail*. This log also seems to have missing records. For instance, most of the activities do not have the *complete* event logged. Thus, the PPA approach did not obtain any results for those activities, as PPA requires at least start and complete events logged in order to obtain meaningful results. We refer to this log as the *Incident* log. We use the inductive miner technique [12] in order to obtain the process models from the logs. We show below experimental results for those logs.

We chose financial process model that are generated from the inductive miner technique in order to calculate case effective time. We build RPST for this model. So, according to RPST we obtain effective time for case *17366* which is equal to 32 minutes. For more details about how we obtained this result see this link[8] .

We will show some results about combined dimensions. Table 8 summarizes the results of the combined activity/case measures. For the *Financial* log, we found that there is no difference in effective time between PAP and our approach because the log contains only start and complete events without suspend in between. In BPA, there are differences in values because BPA computes service time as the difference between start and complete events. In our approach, we compute it as a difference between allocate and complete events. With respect to the waiting time measure, there is a difference between our approach and BPA. The difference is due to that BPA computes the waiting time as the difference between *allocate* and *start* events and also, between *start* and the latest *complete* event, if activity is restarted with no allocate. There are also differences with respect to the *Incident* log. For service time, BPA computes it as the summation of the differences between *complete* event and the preceding *complete* event, in case no *start* event is logged, and the difference between *complete* event and the very first *Start* event even if it is not for the same activity. In our approach, if the activity has no start time we compute service time as the difference between *complete* event and the

[6] https://tinyurl.com/FinacialLog
[7] https://tinyurl.com/IncidLog
[8] https://www.dropbox.com/sh/n5e5wqmu4rymxmw/AAB77woc6mLChByDf4s1QuKKa?dl=0

preceding *complete* event. For waiting time, there is no result for all approaches due to missing events.

Table 9 summarizes the results of the resource/case dimensions for both logs. For the *Financial* log, for the waiting time measure, BPA computes that measure in the same way as activity/case dimensions. Our approach computes it as the difference between *allocate* and *start* events even if it is with a different resource. For the *Incident* log, for the service time measure, the difference is due to that BPA computes it as the difference between *complete* event and the very first *start* event, if there is no *complete* event in between them. Our approach computes it as the summation of the differences between *start* event and the next *start* event. With respect to waiting time, BPA computes it as the difference between *start* event and the immediately preceding *Start* events.

Table10 summarizes the results of combined resource/ activity dimensions. There are differences for both logs for service time and waiting time due to the same reasons highlighted above.

Measures	**Logs**	**Activity/Case**	**PAP**	**BPA**	**Our Approach**
Effective Time	Financial Log	w_completerean_anvraag / 173703	24.3 minutes	–	24.3 minutes
Service Time		w_completerean_anvraag /173703	–	24.3 minutes	135.7 minutes
Waiting Time		w_completerean_anvraag / 173703	111.5 minutes	171.5 minutes	111.5 minutes
Service Time	Incident Log	Completed /1-364285768	–	1110746.5 minutes	1102121 minutes

Table 8. Results of the combined activity/case dimensions

	Logs	Resoruce/Case	BPA	Our Approach
Service time	Financial log	11201/173694	34 minutes	34 minutes
Waiting time		11201/173694	120.3 minutes	10262 minutes
Service time	Incident Log	Anne Claire/1-364285768	8625.5 minutes	1042688 minutes
Waiting time		Anne Claire/1-364285768	37071.8 minutes	1040043 minutes

Table 9. Results for resource/case dimensions

	Logs	**Resource / Activity**	**BPA**	**Our Aproach**
Service Time	Financial Log	10609 / W_Nabellen offerets	14684.5 minutes	14698 minutes
Waiting Time		10609 / W_Nabellen offerets	275776.3 minutes	13.5 minutes
Service Time	Incident Log	Eric / Completed	345401.6 minutes	83241 minutes

Table 10. Results of combined resource/activity dimensions

5 Related Work

There are several existing approaches that measure performance of business process from event logs. The work by van der Aalst and van Dongen [2] further extended in [9] presents an approach that extracts performance information by replaying a timed event log onto work flow nets. Based on that more information related to time dimension and flexibility dimension can be obtained such as synchronization time of specific place and also knowing the probability for a specific path on the net. These measures mainly depend on process perspective. Also, there are many metrics that can be obtained directly from event logs related to activity such as waiting time , execution time of specific activity. In our approach, we assume the knowledge about the process model beforehand and we apply our calculations directly on the log without the need to mine the process first.

In [1], the authors used an alignment technique in order to know the relationship between a process model and a log. This technique is used in checking conformance and analyzing performance. Based on replaying techniques, many kinds of analysis techniques can be obtained, by annotated information about average execution time of activities on process model and also any other general information. Based on that we can obtain more information about case perspective. In [10][13],presented an approach that quantify the relationship between workload and processing speeds (i.e,service time) based on linear regression techniques. Mainly, it depends on resources perspective in order to know more information about why resources taking more time in doing his/her work. Compared to our approach, we can obtain measures for other perspectives, e.g. case perspective, and we can obtain more detailed measures by combining the three perspectives.

[22] proposes a technique that extracts performance metrics from event logs. Processes are defined using a formal specification languages and then the traces, log, are examined against that specification for deviations and for measuring performance measures such as the maximum duration of a process over all its traces, instances. It is not clear which perspectives are covered in this approach as measures were discussed by example. Apparently, the notion of effective time is not supported by that approach. Moreover, the awareness of parallelism in the case perspective is not present.

[4,7] proposed an approach that extracts performance information from discovered model and event logs. Using replaying technique onto discovered model in order to obtain more metrics about process such as node waiting time. Also, there are more metrics that can be obtained directly from event log such as number of cases. This approach mainly depend on activity, case and process perspectives. Compared to our approach, we extract more advanced measures for the resource, case and activity perspectives and their combinations. Moreover, for the case perspective, using the process knowledge, we can get more accurate results that would not be obtained by examining the log only.

In [25], the authors presented an approach that extracts performance information from event logs based on the concept of event gaps analysis. This approach takes into account both case and resource perspectives to tackle measures of performance such as getting daily/weekly working times, waiting times and also workload for each resource. This approach does not consider the original process model within its analy-

sis. Compared to our approach, we can get more accurate results especially for the case perspective, case effective time.

6 Discussion

In this paper, we have presented an approach to extract time-related performance indicators from process execution logs. In addition to the log, the approach requires the knowledge of the process model and its RPST to calculate accurate measures. The execution assumes that events are generated with respect to the activity life cycle where each state transition on an activity lifecycle is reflected in the log. The approach supports four measures, namely effective time, service time, waiting time and sojourn time. These can first be calculated at the finest granularity of the combination of a case, an activity and a resource. Later on, several aggregations can be applied to obtain more aggregate measures. We showed that for the case perspective, the aggregate effective time measure is not trivial and requires knowledge about the process model.

We assume that logs are information rich, with respect to the level of detail reported about activity lifecycle events. However, with poorer logs, our approach is still able to report measures with more accurate values as has been shown in Section 4. At least the *start* and *complete* events must be recorded in order for our approach to work, which is the case with most of the logs. Yet, Looking at centralized execution environment, it is however possible to have that level of detail per state transition recorded in the log and the strength of our approach can be better observed.

Our approach depends on deriving RPSTs to analyze the relationship between the different activities in order to obtain case measures. One limitation here is the degree of unstructuredness of the input process model. If parts of the process are rigids, i.e., unstructured parts, the accuracy of the measures will be affected. However, in some cases, the process can be restructured to get rid of the rigids as shown in [19,16,15]. Yet, this remains as a limitation that we are aim to address in future work.

We have showed also the possibility to obtain combined measures between, e.g., a case and a resource. There are, however, possibilities to extract more complex measures. For instance, one would be interested to see correlation of resources performance over a case or a collection of cases, that belong to the same process model or to different models. These and other directions are considered as further directions for future work.

Last but not least, we intend to apply our approach within the context of an actual process execution environment in order to better evaluate our approach.

References

1. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
2. van der Aalst, W.M.P., van Dongen, B.F.: Discovering workflow performance models from timed logs. In: EDCIS 2002. LNCS, vol. 2480, pp. 45–63. Springer (2002)
3. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: Prom: The process mining toolkit. In: (BPMDemos 2009). CEUR Workshop Proceedings, vol. 489. CEUR-WS.org (2009)

4. Adriansyah, A.: Performance Analysis of Business Processes from Event Logs and Given Process Models. Master's thesis, Technische Universiteit Eindhoven (August 2009)
5. Ballard, C., Mcdowell, S., White, C.: Business Performance Management Meets Business Intelligence. IBM (July 2005)
6. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in {BPMN}. Information and Software Technology 50(12), 1281 – 1294 (2008)
7. van Dongen, B.F., Adriansyah, A.: Process mining: Fuzzy clustering and performance visualization. In: BPM Workshops. LNBIP, vol. 43, pp. 158–169. Springer (2009)
8. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
9. Hornix, P.T.: Performance Analysis of Business Processes through Process Mining. Master's thesis, Technische Universiteit Eindhoven (January 2007)
10. Joyce Nakatumba, Wil M. P. van der Aalst: Analyzing resource behavior using process mining. In: BPM 2009 Workshops. LNBIP, vol. 43, pp. 69–80. Springer (2009)
11. Kimball, R., Reeves, L., Thornthwaite, W., Ross, M., Thornwaite, W.: The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses with CD Rom. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1998)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM 2013 Workshops. LNBIP, vol. 171, pp. 66–78. Springer (2013)
13. Nakatumba, J.: Resource-Aware Business Process Management: Analysis and Support. Ph.D. thesis, Technische Universiteit Eindhoven (December 2013)
14. Parmenter, D.: Key Performance Indicators: Developing, Implementing, and Using Winning KPIs. Wiley, second edn. (2010)
15. Polyvyannyy, A.: Structuring Process Models. Ph.D. thesis, University of Potsdam (2012)
16. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. In: BPM 2010. LNCS, vol. 6336, pp. 276–293. Springer (2010)
17. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Inf. Syst. 37(6), 518–538 (2012)
18. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. Comput. J. 57(1), 12–35 (2014)
19. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling hidden unstructured regions in process models. In: OTM 2009. LNCS, vol. 5870, pp. 340–356. Springer (2009)
20. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: WS-FM 2010. LNCS, vol. 6551, pp. 25–41. Springer (2010)
21. Polyvyanyy, A., Weidlich, M.: Towards a compendium of process technologies - the jbpt library for process model analysis. In: CAiSE 2013 Forum. CEUR Workshop Proceedings, vol. 998, pp. 106–113. CEUR-WS.org (2013)
22. Popova, V., Sharpanskykh, A.: Formal analysis of executions of organizational scenarios based on process-oriented specifications. Appl. Intell. 34(2), 226–244 (2011)
23. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer (2005)
24. Singh, S., Woo, C.: A Methodology for Discovering Goals at Different Organizational Levels. In: (BUSITAL) 2008. CEUR Workshop Proceedings, vol. 336, pp. 16–30. CEUR-WS.org (2008)
25. Suriadi, Suriadi and Ouyang, Chun and van der Aalst, Wil M.P. and ter Hofstede, Arthur H.M : Event gap analysis : understanding why processes take time. Tech. rep., School of Information Systems, Science & Engineering Faculty, Queensland University of Technology (2014)

26. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. 68(9), 793–818 (2009)
27. Velimirovi, D., Velimirovi, M., Stankovi, R.: Role And Importance Of Key Performance Indicators Measurement. Serbian Journal of Management 6(1), 63–72 (2011)
28. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)