Extracting Accurate Performance Indicators From Execution Logs Using Process Models

Nesma M. Zaki*, Ahmed Awad†, Ehab Ezat‡
Information Systems Department
Faculty of Computers and Information
Cairo University, Egypt
Email: * n.mostafa@fci-cu.edu.eg, † a.gaafar@fci-cu.edu.eg, ‡ 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, business goals**

I. 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. Measuring the level of fulfillment of these goals is difficult as goals are unquantifiable measures. Thus, organizations define Performance Indicators (PIs) [1] as quantifiable measures whose assessment reflects the level of achievement of goals. PIs are aggregated measures about the transactional level performance [2], e.g., service time, workload 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 [3]. Thus it makes sense to evaluate the organizational performance based on the performance of its running business processes. Enacted business processes produce execution logs. An execution log carries details about the activity that have been executed, its timestamp, human resource, data that have been processed and the process instance in which the activity has been executed. The quality of the log depends on the level of maturity and automation of the process execution within the organization.

Logs contain the four perspectives which are the process, the case, i.e process instance, the activity and the resource. In addition, logs contain information about the lifecycle state of activities, e.g., start, suspend, resume, complete, etc. From logs, we are able to measure PIs for each of these perspectives or their combinations such as resource/activity, activity/case or case/resource. For example, cycle time and workload of resources are among PIs that can be extracted from execution log. Also, these PIs can be defined in different dimensions which are time, cost, quality and flexibility dimensions[4]. Most of the available approaches to extract PIs from logs [5], [1] address poor logs. That is, logs that expose at most events for the *start* and *complete* of activities or just the complete event.

In this paper, we study the extraction of PIs from logs that expose more about activities lifecycle. That is, more events other than start and complete are recorded. These are (★★★★) or higher logs of quality according to [6]. These logs are generated from an automated execution of the processes, i.e., via a central execution engine. 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 metrics that help in the performance analysis. 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 II overviews our approach and briefly discusses some of the background concepts and techniques that are used throughout the paper. Section III presents our contribution in measuring PIs from process execution logs. In Section IV, we evaluate the proposed approach against other approaches. Related work is discussed in Section V. A critical discussion of the approach and an outlook on future work is presented in Section VI.

II. Overview & Background

We measure four types of time-based PIs. These are *service*, *effective*, *waiting* and *sojourn time*. These PIs are measured for four perspectives, namely: process, case, activity and resource and any of their combinations.

The *service time* measure defines the time elapsed between the creation time of an object,e.g. case, a resource or an activity, in execution environment and its termination time. 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 *offering* an activity and the time it has been terminated. This measure applies to the combination of activity and case perspectives only. The *effective time* measure excludes from the service time those intervals where the object was idle.

Figure 1 summarizes how 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. [7]. Figure 2 summarizes the lifecycle we support in this research. 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) [8] is an appropriate abstraction that provides this information. The pre-processing step in Figure 1 is to obtain the RPST and makes it ready for further steps of performance measures calculation.

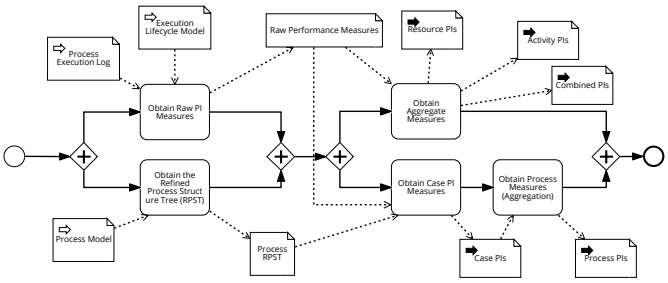

Figure 1: Overview of the approach

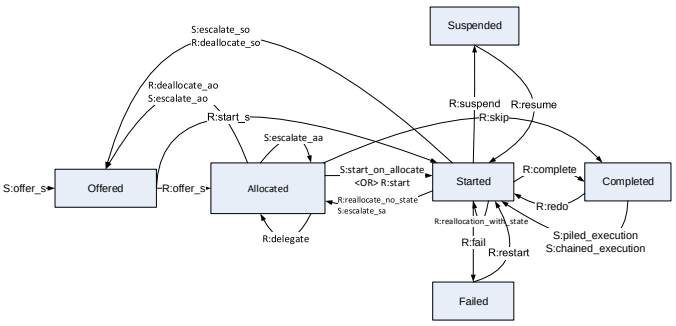

Figure 2: Task life cycle adapted from [7]

The first processing step is to compute what we call the *raw performance measures RW*. 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 perspectives 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 [9]. In our case, the raw performance measure stores the value of the measure per all combinations of the perspectives values. In our case, the perspectives 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 effective, service and waiting time respectively,
- $start$ 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 the *three* raw performance measures that 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 perspectives, e.g., resource, activity, or case, we need to apply aggregations over members of the RW. For resources and activity perspectives, 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 their individual effective time within the case. Rather, the effective time 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.

A. 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 refined process structure tree (RPST) [8], 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 are organized into a hierarchy according to the nesting relation, the process structure tree. Each fragment can be further categorized as either *sequence*, *xor*, *or*, *and*, *loop* or *rigid*. 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 [10] may restructure un/semi-structured process models. However, it has not been proven that it will always restructure it.

B. Running Example

Figure 3 describes an order fulfillment process adapted from [3]. The process starts when a seller receives an order. The seller checks the availability of the product in the stock. If the product is in stock, it will be retrieved. 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 the 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 I is part of an execution log that is generated from the that process.

Table I: Sample event log for the 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
⋮	⋮	⋮	⋮	⋮	⋮

III. Calculating Performance Measures

As stated earlier, we define four measures of time-based PIs which are effective time, service time, waiting time and sojourn time with the different perspectives discussed before, resource, activity, case, process and their combinations.

A. 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. 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 1 illustrates how to calculate Effective time raw performance measure.

To explain how the algorithm works, assume that the input log L to the algorithm is Table I 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, $C = \{1, 2, 14\}$. 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

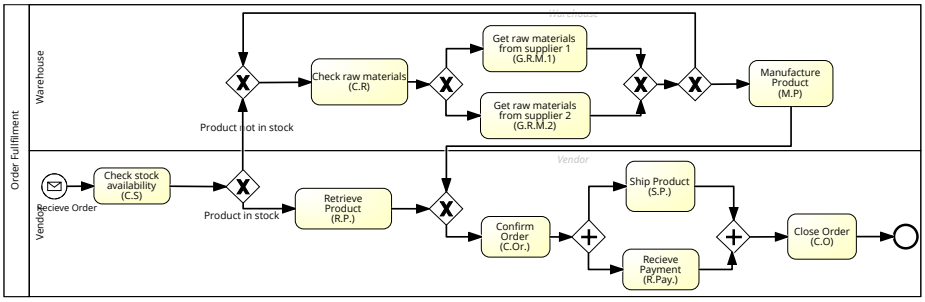

Figure 3: Order Fulfillment

Table II: Raw performance measures for the log from Table I

#	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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Algorithm 1 Calculate 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;

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, 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 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.

To give an example, we calculate effective time for *Kareem* according to activity *S.P* in case 1. As shown in Table I, *Kareem* started this activity at time $t_s = 18-03-2015\ 09:28$ 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)+ ($t_{e'}$ - $t_{s'}$). This information is stored in RW as seen in Table II.

1) Case Effective Time: Calculating *case* effective time depends on RW and also on the RPST of the model P. Algorithm 2 computes case effective time.

Algorithm 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

Algorithm 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: $\Omega = \sigma_{a,\perp,c,E}(RW)$ ▷set of selected elements from 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** $type(RPST.r) = \{and, or\}$ **then**
9: $CaseEffectiveTime+ = getMaxEffectiveTime(RPST.r)$
10: **else if** $type(RPST.r) \in \{seq, xor, loop, rigid\}$ **then**
11: $CaseEffectiveTime+ = getSumEffectiveTime(RPST.r)$
12: $CRW = CRW \cup (p, c, CaseEffectiveTime)$

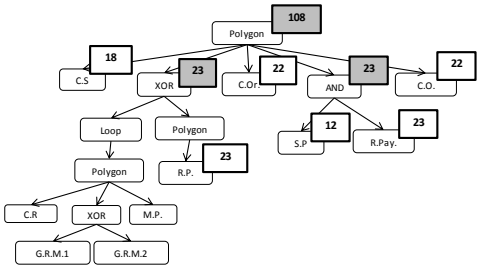

Figure 4: RPST of Figure 3 instantiated for case 1

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 types *and* or *or* , the function $getMaxEffectiveTime$ is invoked over children of r. In case of *or* block, if one of the branches is not activated, we represent it as 0. If the node of is other types than *and*, *act* and *or* 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* or *or* blocks cases, 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, is updated with the value. 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.

B. 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.

Algorithm 3 illustrates how to calculate service time raw performance measure. The input of this algorithm is the same as in the Algorithm 1.

Service time is computed as the time elapsed between a) resource r *started* working on the activity a and the time he has *completed* the activity, or b) *allocate* time of work of r on a and the time at which r *completed* 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 27, either by *allocating* or *starting*, Line 29, a to some other resource. At the times of *completion or failure*, the service time raw performance measure is updated, Line 18. 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 *service time* is added to the raw performance measure, Line 32.

Algorithm 3 Calculate service time raw performance measure

Input: A process Model identifier p
Input: Execution log L for a specific process model p
Output: RW The service time raw performance measure

$T_s = null$ ▷start of work timestamp
$T_e = null$ ▷end of work timestamp
$T_{e'} = null$ ▷end of work timestamp for another resource
$flag = true$; $occurrence = 0$
$C = CASE(L)$ ▷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; $flag = true$; 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** ($j.event\_type \in \{started, allocated\}$ and $flag = true$) **then**
12: $T_s = j.timestamp$;
13: $flag = false$
14: **if** ($X(j).event\_type \in \{completed, failed\}$) **then**
15: $T_e = j.timestamp$
16: **if** ($T_s \neq null$) **then**
17: $Ser\_time = Ser\_time + (T_e - T_s)$
18: $RW = RW \cup \{(p, c, a, r, occurrence, 'S', start, end, Ser\_time)\}$
19: occurrence = occurrence + 1;
20: $flag = true$;
21: **else if** (T_s is null) **then**
22: $t = getPreviousCompleteTimestamp(c, 'completed')$
23: $Ser\_time = Ser\_time + (t - T_e)$
24: $RW = RW \cup \{(p, c, a, r, occurrence, 'S', start, end, Ser\_time)\}$
25: occurrence = occurrence + 1;
26: $flag = true$;
27: **else if** ($j.resource \neq r$) **then**
28: **if** ($T_s \neq null$) **then**
29: **if** ($j.event\_type \in \{started, allocated\}$) **then**
30: $T_e = j.timestamp$
31: $Ser\_time = Ser\_time + (T_e - T_s)$
32: $RW = RW \cup \{(p, c, a, r, occurrence, 'S', start, end, Ser\_time)\}$
33: r = j.resource;
34: **if** ($j.event\_type == completed$ and T_s is null) **then**
35: $T_{e'} = j.timestamp$
36: $Ser\_time = Ser\_time + (T_e - T_{e'})$
37: $RW = RW \cup \{(p, c, a, j.resource, occurrence, 'S', start, end, Ser\_time)\}$

Noisy logs: In case of missing entries for some tasks in the log we are still able to extract performance indicators. In case of a resource r *completed* a task a and there is no *start* event logged, then we compute the service time of a task a for the resource r as difference between *complete* event and preceding *completed* event, Line 23. This information is added to the raw performance measure, Line 24. If resource r *completed* the task a and after a while the task a was *completed* by another resource $r1$ with no *start* event logged in between, then we can compute service time of the task a for the resource $r1$ as difference between *completed* event for $r1$ on a and previous *completed* event for r on a , Line 36. This information is added to the raw performance measure, Line 37. We will show some results in the evaluation section.

Algorithm 3 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 19 and 25.

We illustrate by an example how to calculate service time raw performance measure for combination of resource, case and activity. For example, in Table I, we find that *Marwan* allocated activity *G.R.M.2* for case 14 at time t_s = 08 − 04 − 2015 03 : 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 t_{comp} - t_s whereas as the effective time is just 19 hours.

1) Case Service Time: is defined as the time elapsed from the start of the case and its end. For example, cf. Table II, 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.

2) Activity Service Time: The calculation of *activity* service time can be done by first applying an activity selection on *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.

C. 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 I, 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.

D. 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 I, 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 II in rows 4 and 5.

IV. Evaluation

We have implemented the algorithms [1] presented in Section III using Java and relational databases. Also, we have used the jBPT library [11] 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 fragment types as loops, xor, or, and, sequence or rigid. The complexity of the algorithms is $O(C*A*N^1)$ where C is the number of distinct case identifiers in the log and A is the number of distinct activity identifiers in the log and N is the size of the log.

We found three approaches that analyze process performance which are the Performance Analysis with Petri net (PAP)[12][2], Basic Performance Analysis (BPA)[3] and event gap analysis [13]. All are implemented as ProM [14] 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. Table III compares and clarifies the features of our approach against the other approaches.

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

Table III: Feature comparison between three approaches

Performance Analysis with Petri Net (PAP): The main objective of this approach is to extract performance information from event logs and process models, 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 .

A. 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 resources information as well as more events to reflect transition in activities lifecycles. We compare PAP and BPA plugins to our approach using the complete log from Table I. We generated the Petri net underlying the order fulfillment model based on transformation rules in [15] as it was required by the plugin.

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

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

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

Table IV summarizes results of aggregated service and waiting time of the combined resource/activity perspectives. We applied these measures for resource *Marwan* within activity get raw materials from supplier 2 (G.R.M.2). The difference in values between these approaches with respect to execution time is due to that the *BPA* computes it 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 I, 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 I, 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. We have done more evaluations using different combined measure like resource/case. Due to space limitations, these results are omitted, more details and results can be found in [16].

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 IV: Results of the combined resource/activity perspectives

B. Real Logs

We chose two real logs from the BPI Challenges. One of these logs is taken from a Dutch Financial institute [4] and contains data that represent a process of personal loans applications. It contains 13087 cases with a total of 26200 events. This log exposes three event types, *Schedule, Start* and *Complete*. We refer to this log as the *Financial* log.The other log is taken from Volvo IT Belgium and contains data that represent the IT incident management system [5]. It contains 7554 cases with a total of 65533 events. It exposes event types, *In call, Cancelled, Unmatched, Wait, In progress, Awaiting assignment, Assigned, Resolved* and *Closed*. We mapped these event types to match our lifecycle. Most of the activities do not have the *complete* event logged. Thus, the PAP approach did not obtain any results for those activities, as PAP 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 [17] in order to obtain the process models from the logs. We show below experimental results for those logs.

Table V 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. For the service time measure, the differences between the values is due to that BPA considers only difference between the timestamps of *started* and *completed* events. Our approach computes it as the difference between the timestamps of *allocated* and *completed* 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 the service time measure,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 preceding *complete* event. For the effective and waiting time measures, there is no result for all the approaches due to missing events.

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 V: Results of the combined activity/case perspectives

Table VI summarizes the results of the resource/case perspectives for both logs. For the *Financial* log, for the waiting time measure, BPA computes that measure in the same way as activity/case perspectives. 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. We have done more evaluations using different combined measure like resource/activity. Due to space limitations, these results are omitted, more details and results can be found in [16].

	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 VI: Results for resource/case perspectives

V. 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 [5] further extended in [12] presents an approach to extract performance information by replaying the log onto work flow nets. These measures address the process perspective. Also, there are many metrics that can

[4]https://tinyurl.com/FinacialLog
[5]https://tinyurl.com/IncidLog

be obtained directly from event logs related to activity such as waiting time, execution time of specific activity.

In [18], the authors used a conformance checking technique that can analyze performance. By replaying the log, average execution time of activities on process model can be obtained with respect to the case perspective. Similarly as the approach in [4]. [19] and [20] present an approach that quantifies the relationship between workload and processing speeds (i.e,service time) based on linear regression techniques. It measures mainly the resources perspective. Compared to our work, we can obtain measures for other perspectives, e.g., case perspective, and we can obtain more detailed measures by combining perspectives.

In [13], 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 analysis. Compared to our approach, we can get more accurate results especially for the case perspective, case effective time.

We claim that we obtain more accurate measures than existing approaches. In our approach, the awareness of the parallelism in the case perspective is present. Also, our approach supports more transitions than other approaches not only start and complete events.

VI. Discussion

In this paper, we have presented an approach to extract time-related PIs 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 in the log are generated with respect to activity lifecycle. This approach supports four measures, namely effective , service , waiting 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 mainly depend on deriving RPSTs in order to obtain case measures. However, the limitation here is the degree of the 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 as we sum up the values. Also, we did not take into consideration when calculating PIs the working hours only of the resource ignoring holiday and weekends. Yet, we intend to address these limitations in future work.

We assume that logs are information rich, with respect to the level of detail reported about activity lifecycle events. However, with poorer logs, we are still able to produce measures that are at least as accurate as other approaches.

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

References

[1] D. Parmenter, *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs*, 2nd ed. Wiley, 2010.

[2] C. Ballard, S. Mcdowell, and C. White, *Business Performance Management Meets Business Intelligence*. IBM, July 2005.

[3] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[4] A. Adriansyah, "Performance analysis of business processes from event logs and given process models," Master's thesis, Technische Universiteit Eindhoven, August 2009.

[5] W. M. P. van der Aalst and B. F. van Dongen, "Discovering workflow performance models from timed logs," in *EDCIS 2002*, ser. LNCS, vol. 2480. Springer, 2002, pp. 45–63.

[6] W. M. P. van der Aalst et Al., "Process mining manifesto," in *BPM Workshops*, ser. LNBIP, vol. 99. Springer, 2011, pp. 169–194.

[7] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *CAiSE 2005*, ser. LNCS, vol. 3520. Springer, 2005, pp. 216–232.

[8] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree," in *WS-FM 2010*, ser. LNCS, vol. 6551. Springer, 2010, pp. 25–41.

[9] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, and W. Thornwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses with CD Rom*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[10] A. Polyvyanyy, L. García-Bañuelos, D. Fahland, and M. Weske, "Maximal structuring of acyclic process models," *Comput. J.*, vol. 57, no. 1, pp. 12–35, 2014.

[11] A. Polyvyanyy and M. Weidlich, "Towards a compendium of process technologies - the jbpt library for process model analysis," in *CAiSE 2013 Forum*, ser. CEUR Workshop Proceedings, vol. 998. CEUR-WS.org, 2013, pp. 106–113.

[12] P. T. Hornix, "Performance analysis of business processes through process mining," Master's thesis, Technische Universiteit Eindhoven, January 2007.

[13] S. Suriadi, C. Ouyang, W. M. van der Aalst, and A. H. ter Hofstede, "Event Gap Analysis : Understanding Why Processes Take Time," QUT: ePrints, Tech. Rep., 2014.

[14] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, and T. Weijters, "Prom: The process mining toolkit," in *(BPMDemos 2009)*, ser. CEUR Workshop Proceedings, vol. 489. CEUR-WS.org, 2009.

[15] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in {BPMN}," *Information and Software Technology*, vol. 50, no. 12, pp. 1281 – 1294, 2008.

[16] Nesma M. Zaki, Ahmed Awad, and Ehab Ezat, "Extracting Accurate Performance Indicators From Execution Logs Using Process Models," Cairo University, Tech. Rep., 2015. [Online]. Available: http://scholar.cu.edu.eg/?q=ahmedawad/files/extractingaccuratemeasures.pdf

[17] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM 2013 Workshops*, ser. LNBIP, vol. 171. Springer, 2013, pp. 66–78.

[18] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[19] Joyce Nakatumba and Wil M. P. van der Aalst, "Analyzing resource behavior using process mining," in *BPM 2009 Workshops*, ser. LNBIP, vol. 43. Springer, 2009, pp. 69–80.

[20] J. Nakatumba, "Resource-aware business process management: Analysis and support," Ph.D. dissertation, Technische Universiteit Eindhoven, December 2013.