Time-bounded Statistical Analysis of Resource-constrained Business Processes with Distributed Probabilistic Systems

Ratul Saha[1], Madhavan Mukund[2], and R. P. Jagadeesh Chandra Bose[3]

[1] National University of Singapore, Singapore, ratul@comp.nus.edu.sg
[2] Chennai Mathematical Institute, India, madhavan@cmi.ac.in
[3] Xerox Research Center India, India, jcbose@gmail.com

Abstract. Business processes often incorporate stochastic decision points, either due to uncontrollable actions or because the control flow is not fully specified. Formal modeling of such business processes with resource constraints and multiple instances is hard because of the interplay among stochastic behavior, concurrency, real-time and resource contention. In this setting, statistical techniques are easier to use and more scalable than numerical methods to verify temporal properties. However, existing approaches towards simulation techniques of business processes typically rest on shaky theoretical foundations. In this paper, we propose a modular approach towards modeling stochastic resource-constrained business processes. We analyze such processes in presence of commonly used resource-allocation strategies. Our model, Distributed Probabilistic Systems (DPS), incorporates a set of probabilistic agents communicating among each other in fixed-duration real-time. Our methodology admits statistical analysis of business processes with provable error bounds. We also illustrate a number of real-life scenarios that can be modeled and verified using this approach.

1 Introduction

In recent years, a plethora of models have been proposed in the area of Business Process Management (BPM) [3, 10]. These models have been used to analyze large processes from diverse industry sectors such as Internet companies, health care, and finance services. The tasks in these processes are typically mapped to a finite set of shared resources whose allocation depends on a variety of practical constraints. In addition, each process is often replicated as a large number of instances. To optimize performance, one needs to be able to analyze resource-constrained business processes with well-defined confidence in the result.

BPM systems often do not realize deterministic behavior and incorporate stochastic decision points. This is due to both the increasing complexity of such systems, which makes the exact control flow difficult to capture, as well as the inclusion of uncontrollable components in business processes. Even when the probability distributions can be measured or approximated from domain knowledge or historical data [2], model-based analysis [3] of such systems is hard due

to the interplay between stochastic behavior, concurrency, time taken to perform tasks, and resource contention.

We propose a novel modular approach towards modelling resource-constrained BPM (rcBPM) systems. Such systems have a finite set of resources allocated across replicated instances of a stochastic business process. A business process is a set of tasks with logical and temporal dependencies. Each task is mapped to one of the available resources that can perform the task. Resources are assigned following a predetermined allocation strategy. Each task has an execution time, ideally drawn from a probability distribution. For simplicity, we assume the time taken by a task to be fixed—say the mean value of the distribution.

A *case* is an instance of the process model. Multiple cases run in parallel, sharing the same set of resources. Cases need not start simultaneously. We study systems with a fixed number of cases arriving within a given period of time. The cases may follow an arrival process such as a Poisson process.

An example of an rcBPM system is the loan/overdraft process in a financial institution, where cases correspond to applications from different customers. Tasks in the process may include 'submitting', 'reviewing', 'accepting' or 'declining' the application.

We observe that tasks and resources can be considered as individual agents that behave independently and communicate among each other. There are two types of communications among them: (i) task-task interaction, where a completed task passes the thread of control to some other tasks, and (ii) task-resource interaction, which describes the allocation of a task to a resource.

This observation leads us to introduce *Distributed Probabilistic Systems (DPS)* to model rcBPM systems. A DPS consists of a set of communicating probabilistic agents. Each agent has a finite local state space. Periodically, agents synchronize with each other on common *actions* to perform joint probabilistic *events*. Each event has a fixed duration and cost. A scheduler is used to resolve non-determinism: if an agent can take part in more than one synchronization at a global state, the scheduler determines which one of them is to be performed.

In general, after the scheduler resolves non-determinism, multiple independent actions are enabled at a global state of a DPS, which can be executed concurrently. Each synchronization action among a set of agents leads to an event being chosen according to a probability distribution. Each event has a fixed duration, after which the local states of the participating agents change. We show that the dynamics of a DPS with a fixed scheduler can be captured as a discrete-time Markov chain. The DPS model can be viewed as a Markov Decision Process (MDP) variant of Distributed Markov Chains (DMC) [20].

We model an rcBPM system as a DPS. Each task is an individual agent incorporating the states of a task, such as 'ready to perform', 'waiting for a resource', 'busy executing', 'finished' etc. When a task finishes, it triggers other task agents in the control flow that are ready to perform. Each resource is also a simple agent, looping between being 'available' and 'busy'. The scheduler maps each task waiting for a resource to an available resource. This results in a synchronization that generates an event whose duration captures the time taken to

perform the task. We also model the start and end states of a process as agents, to model the arrival and completion of a case.

To verify temporal properties of rcBPM systems, we use Statistical Model Checking (SMC) [14, 24] . A typical property is of the form "when C cases arrive with constant arrival density λ, at least $x\%$ of the cases will complete within time t, with probability at least p". Since the DPS model of an rcBPM system can be viewed as a discrete-time Markov chain, we can simulate an rcBPM system and use hypothesis testing to verify properties with provable error bounds.

We illustrate our approach using a business process [7] depicting loan/overdraft applications of a large financial institution. The process has been mined from real-life event logs from BPM Challenge 2012 [22]. The process has 46 resources for performing 15 tasks in the process. We scaled up to 500 cases arriving at the rate of 1 case every 10 seconds. We show (i) the relationship between the number of cases arriving within a fixed time bound and the fraction of cases that complete, and (ii) the relationship between the minimum time of completion and the fraction of cases completed for a fixed number of total cases.

To summarize, our contribution is as follows: (i) we propose a model of distributed probabilistic systems where events are assigned time and cost values, and demonstrate that the model acts as a Markov chain for a well-behaved class of schedulers, (ii) we provide a strong theoretical foundation for resource-constrained business processes modelled as distributed probabilistic systems, (iii) we demonstrate a statistical model checking based technique for inferring time-bounded properties of business processes with multiple cases and a finite set of shared resources. To the best of our knowledge, this is the first attempt to provide a simulation based technique for analyzing resource-constrained business processes with provable error bounds and sound sample size analysis.

The paper is structured as follows. Section 2 introduces resource-constrained BPM systems and different properties of interest. Section 3 defines the Distributed Probabilistic System (DPS) model and a statistical model checking technique for DPS. Sections 4 and 5 provide a proof-of-concept demonstration of our approach. Section 4 describes how rcBPM systems are modelled using DPS. Section 5 discusses experimental results for a simple example. Finally, Section 6 provides a summary and future directions.

Related Work The need for an underlying formal model for business processes has been long felt. Workflow nets (WF nets, a class of Petri nets), equipped with clean graphical notation and abundance of analysis techniques, have served as a solid framework for BPM systems [1, 6, 4].

For this discussion, we restrict to related work that involves modelling or analyzing stochastic business processes. The most prominent modelling formalism for stochastic analysis for business processes is (generalized) stochastic WF nets [18, 9]. Such a system is modelled using exponentially distributed firing delay with timed transitions. A few recent papers [21, 15, 8] also demonstrate a generic Markovian analysis that is mostly applicable to rigidly structured WF systems. The work of [12] focuses on modelling BPM systems as Markov decision processes in the language of PRISM [13]. These works are either very simplistic

in nature, where block-like patterns are chained together, or hard-to-tackle models involving arbitrary nondeterminism that cannot be readily adopted for sound simulation techniques. Most importantly, extending these approaches to model business processes with shared resources across multiple cases is not obvious.

Analyzing resource-constrained BPM systems with simulation-based techniques is not new [16, 17], but rigorous statistical analysis has often been limited to computing analysis of variance and confidence intervals. The deductions are often difficult to justify and can be arbitrarily far from truth—van der Aalst correctly points out that "simulation does not provide any proof" [5]. Our work is the first to (i) provably bound the error of analysis of business processes with finite resources shared across multiple cases arriving at different time points and (ii) provide a sound analysis of the sample size of simulation.

2 Resource-constrained BPM (rcBPM) Systems

A resource-constrained BPM (rcBPM) system consists of two main components: (i) the process, instantiated as a fixed number of cases, and (ii) a finite set of resources. An rcBPM system is then accompanied by a resource allocation strategy. To explain these, we use a process model that has been mined [7] from a real-life event log of loan/overdraft applications of a large financial institution. Along with the process, the average time for each resource to perform a particular task has been mined. The event log is obtained from BPI Challenge 2012 [22].

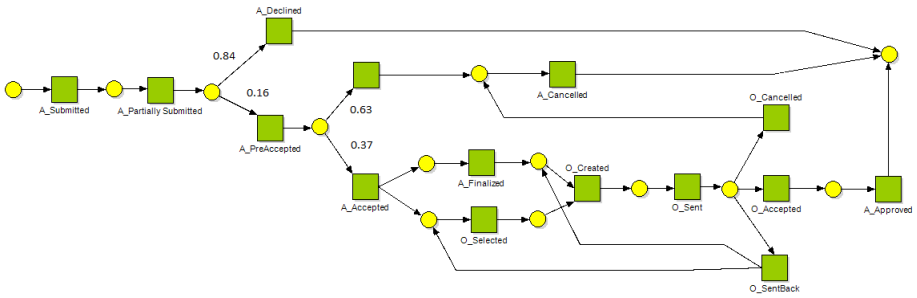


Fig. 1. An example process of an rcBPM system (in Petri net notation).

The process A process in an rcBPM system consists of a start state, a finite set of tasks and an end state. The tasks in the process are combined in sequence or parallel. We assume that the control flow is probabilistic: a discrete probability distribution is attached to each set of outgoing choices and sequential tasks have outgoing probability 1. Each case is an instantiated copy of the process.

The Example Fig. 1 demonstrates, in Petri net notation, a process for loan/overdraft applications of a large financial institution. We consider two sub-processes—namely the application and offer sub-processes—of the overall

loan/overdraft application process. The tasks of the application and offer subprocesses are prefixed with "A_" and "O_" respectively. From historical data, we estimate the probability values of outgoing edges from places in the Petri net. Unmarked edges have probability 1. The process starts with the submission of an application (A_Submitted/A_PartlySubmitted). An application can be declined (A_Declined) if it does not pass any checks. The probability of an application being declined outright is estimated to be 0.84. Applications that pass the checks are pre-accepted (A_PreAccepted) with probability 0.16. Often additional information is obtained by contacting the customer by phone. Based on this information, an application can be cancelled (A_Cancelled) with probability 0.63 or accepted (A_Accepted) with probability 0.37. Once an application is accepted, it is finalized (A_Finalized) and, in parallel, an offer is selected for the customer (O_Selected). An offer is then created (O_Created) and sent to eligible applicants and their responses are assessed (O_Sent). A customer then may (i) accept the offer (O_Accepted) with probability 0.01, (ii) cancel the offer (O_Cancelled) with probability 0.95, or (iii) send the offer back (O_SentBack) with probability 0.04. If an offer is sent back, a new offer is created for the customer. If the offer is accepted, the application is finally approved (A_Approved). Declining, cancellation or approval signals the end of the application.

Resources A finite set of resources is provided, each capable of performing a subset of tasks in the process. Different resources can take different time for performing the same task. We may further group resources with the same behaviour and capability into roles.

In the running example, we profiled the top 46 of the busiest resources from the event log. The resource-task matrix with cell values representing the average time taken by the resource on the particular task is shown (truncated) in Table 1 in the appendix. The columns indicate the resources and the rows indicate the tasks. If a cell is empty, it indicates that the task is not assigned to the particular resource. If the value in a cell is 0, the resource performs the task instantly.

Resource allocation strategy At any moment, multiple tasks can compete for a number of available resources and, conversely, multiple resources may be available for a single task. The resource allocation strategy is responsible for assigning each task to a single resource. A strategy is said to be deterministic if, given the same snapshot of the system, the strategy always picks the same resource allocation for tasks. While our model can be extended to accommodate randomized schedulers, to keep the formalism simple, we focus on deterministic schedulers. For example, the flexible assignment policy [16] is deterministic: given a specialist-generalist ratio, it assigns the most specialist (available) resource to a given task.

In the running example, we assume that a priority based scheduler is available. This scheduler assumes that there is an ordering among cases—one can think of it as different tiers (platinum/gold/silver) of customers. We also assume that given a case, there is an ordering among its tasks. Hence, there is a total ordering among tasks across all cases. For simplicity, we also assume a total order among the resources. Hence at any configuration of the system, the

scheduler goes through the resources in ascending order. For each resource, if a set of assigned tasks are available for that particular resource, it schedules the lowest ranked task among them. We note that such a scheduling policy may not be optimal, but real-life schedulers for business processes are often simple in nature. Also, while our model permits more complex schedulers, we have chosen a simpler one to explain the approach and demonstrate some experimental results.

The modeling formalism and properties of interest

We do not restrict ourselves to any particular modeling formalism and discuss the support of our modeling technique in terms of workflow patterns [19]. We support the core patterns—sequence (WCP-1), parallel split (WCP-2), synchronization (WCP-3), (probabilistic) exclusive choice (WCP-4), structured synchronizing merge (WCP-7). In short, we allow parallel and XOR-splits as well as parallel and XOR-joins as long as each parallel split is matched with a parallel join. With some engineering, our approach can be extended to more unstructured models as long as arbitrarily new thread of controls cannot be spawned. However for this work, we focus on structured business processes.

We are interested in time-bounded temporal properties of rcBPM systems. A fixed number of cases arrive following an arrival process. Each task and resource can have their own time limit. Given a time limit, we are interested to verify linear temporal properties of business processes bounding the probability of error. In the running example, let us assume that we are interested in the following property: when C cases arrive at a rate of λ cases per second, with probability at least p, at least $x\%$ cases are completed within time t. While we can analyze more complex temporal properties, in this work, we focus on reachability properties. Note that we can also find optimal parameter values using binary search. For example, if we are interested in optimizing x, we fix the other values p, t, λ and C and use simulations to do a binary search for the optimal value of x in the range $[0, 100]$.

The state-of-the-art approach for analyzing stochastic business process models is to simulate the system an arbitrary number of times. If $p\%$ of the simulations satisfy the desired property, one claims that the property is true with probability at least p, perhaps with confidence interval estimates of certain parameters. However, such conclusions are dependent on the sample size of the simulation and correctness is not guaranteed. We apply the theory of hypothesis testing and sequential probability ratio test, which formally connects the sample size to the desired margin for error. The conclusions that we draw from our experiments come with guaranteed bounds on the probability of error due to false positives and false negatives. Our strategy is explained in detail in Section 3.

3 The Distributed Probabilistic System (DPS) model

We formulate a model in which a distributed set of agents interact through actions that synchronize subsets of agents. A synchronization results in a probabilistic choice between a set of events, each with its own time duration and cost.

We can then use hypothesis testing for statistical model checking of quantitative properties of such systems.

Definition 1 (Distributed State Space). *A distributed state space over n agents $[n] = \{1, 2, \ldots, n\}$ is a tuple $\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$, such that for each agent $i \in [n]$, S_i denotes its finite set of local states and s_i^{in} denotes its local initial state. We abbreviate $[n]$-indexed sets $\{X_i\}_{i\in[n]}$ as $\{X_i\}$ when the context is clear.*

- *For non-empty $u \subseteq [n]$, $\mathbf{S}_u = \prod_{i\in u} S_i$ denotes the set of joint u-states of agents in u. We denote $\mathbf{S} = \mathbf{S}_{[n]}$ to be the set of* global states.
- *For a u-state $\mathbf{s} \in \mathbf{S}_u$ and $v \subseteq u$, $\mathbf{s}_v$ denotes the projection of $\mathbf{s}$ onto v. We do not distinguish between $\mathbf{S}_{\{i\}}$ and $\mathbf{S}_i$, nor between $\mathbf{s}_{\{i\}}$ and $\mathbf{s}_i$.*

Definition 2 (Events and Actions).

- *An* event *over a distributed state space $\mathcal{S}$ is a tuple $e = (src_e, tgt_e)$, such that $\emptyset \neq loc(e) \subseteq [n]$ specifies the agents that participate in e and $src_e, tgt_e \in \mathbf{S}_{loc(e)}$ denote the source and target $loc(e)$-states of e.*
- *Let Σ denote the set of all events over $\mathcal{S}$. Each event e has a* duration $\delta(e)$ *and a* cost $\chi(e)$*, given by functions $\delta : \Sigma \to \mathbb{R}_{\geq 0}$ and $\chi : \Sigma \to \mathbb{R}_{\geq 0}$.*
- *An* action *over $(\mathcal{S}, \Sigma)$ is a collection of co-located events with the same source state, equipped with a probability distribution. Formally, an action is a pair $a = (E_a, \pi_a)$, where $E_a \subseteq \Sigma$ is such that for each $e, e' \in E_a$, $src_e = src_{e'}$, and $\pi_a : E_a \to [0, 1]$ is a probability distribution. We write $loc(a)$ for the set of agents participating in action a and $src(a)$ for the common source state of the events in E_a. Let A denote the set of all actions over $(\mathcal{S}, \Sigma)$.*
- *We assume that each event belongs to exactly one action. In other words $\Sigma = \bigcup_{a\in A} E_a$ and for each $a, b \in A$ such that $a \neq b$, $E_a \cap E_b = \emptyset$.*
- *For a global state $\mathbf{s} \in \mathbf{S}$, $en(\mathbf{s})$ is the set of actions* enabled *at $\mathbf{s}$. Formally, $en(\mathbf{s}) = \{a \mid src(a) = \mathbf{s}_{loc(a)}\}$.*
- *At a global state $\mathbf{s} \in \mathbf{S}$, a set of enabled actions U is* schedulable *if each agent partipicates in at most one action in U. Formally, $U \subseteq en(\mathbf{s})$ is schedulable if for all $a, b \in U$ such that $a \neq b$, $loc(a) \cap loc(b) = \emptyset$. Note that a schedulable set of actions can be executed concurrently since the agents involved do not interfere with each other. Let $sch(\mathbf{s}) \subseteq 2^{en(\mathbf{s})} \setminus \emptyset$ denote the collection of schedulable sets of actions at $\mathbf{s}$.*

Definition 3 (Distributed Probabilistic Systems). *A Distributed Probabilistic System (DPS) is a tuple $\mathcal{D} = (\mathcal{S}, \Sigma, \delta, \chi, A)$ such that*

- *$\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$ is a distributed state space,*
- *Σ is the set of events over $\mathcal{S}$ with duration and cost functions δ and χ, respectively,*
- *A is the set of actions over $(\mathcal{S}, \Sigma)$.*

A DPS $\mathcal{D}$ evolves as follows. All agents start at their initial state, so the initial global state at time 0 is $\mathbf{s}^{in} = (s_1^{in}, s_2^{in}, \ldots, s_n^{in})$.

Suppose $\mathcal{D}$ is at a global state $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ at time t. A set of schedulable actions $U \in sch(\mathbf{s})$ is chosen from the set of enabled actions. We assume the existence of a scheduler that guides this choice.

The actions in U start concurrently and independently. For each action $a \in U$, an event $e_a = (src_e, tgt_e) \in E_a$ is chosen according to the probability distribution π_a, with an associated duration $\delta(e_a)$ and cost $\chi(e_a)$.

The durations $\{\delta(e_a)\}_{a \in U}$ fix a sequentialization of the events $\{e_a\}_{a \in U}$. In general, there will also be a pending list of partially executed events currently in progress, with their own completion times.

Among the list of pending events, old and new, the events with the earliest time to completion finish first. For each completed event e, the local states of agents in $loc(e)$ are changed to tgt_e, while the states of the agents $[n] \setminus loc(e)$ are unchanged.

This gives rise to a new global state where potentially a new set of actions are scheduled, and we repeat the process of choosing a set of actions to schedule. We have to ensure that the scheduler respects the decisions made earlier, so that all pending events remain compatible with the new choice.

We denote a global configuration of a DPS as a *snapshot*, consisting of a global state and a list of partially executed events, with their time to completion from the current time point.

Definition 4 (Snapshots). *A* snapshot *of a DPS $\mathcal{D}$ is a tuple $(\mathbf{s}, U, X)$ where $U \subseteq en(\mathbf{s})$ and $X = \{(a, e, t) \mid a \in U, e \in E_a, t \in \mathbb{R}_{\geq 0}\}$ such that:*

- *$\mathbf{s}$ is the current global state and $U \in en(\mathbf{s})$ is the set of actions that are currently being performed, which may not have started together.*
- *For each $a \in U$, there is exactly one entry $(a, e, t) \in X$ denoting that event $e \in E_a$ is in progress with time $t \leq \delta(e)$ till completion.*

Let $\mathcal{Y}$ be the set of snapshots. Though $\mathcal{Y}$ is uncountable, a DPS will give rise to a discrete set of reachable snapshots, determined by the duration function δ.

Nondeterministic choices between actions are resolved by a scheduler. At each snapshot, the scheduler picks a schedulable set of actions that are pairwise independent. For consistency, this choice should include all the actions already in progress, but not necessarily new ones.

Definition 5 (Schedulers). *A* scheduler *$\mathcal{G}$ is defined over snapshots as follows. Let $y = (\mathbf{s}, U, X) \in \mathcal{Y}$ be a snapshot. Then $\mathcal{G}(y) \in sch(\mathbf{s})$ is such that $U \subseteq \mathcal{G}(y) \subseteq en(\mathbf{s})$.*

We also note that, in general, it is hard to define independence-respecting schedulers for distributed systems. This is closely related to defining local winning strategies in distributed games [11]. The main complication is that a sequentially defined scheduler must behave consistently across different linearizations

that correspond to the same concurrent execution. However, in a DPS, the durations associated with the events fix a canonical linearization, so there is no need to reconcile decisions of the scheduler across different interleavings.

Once we fix a scheduler, we can associate a transition system associated with a DPS whose states are snapshots.

Definition 6 (Transition System). *Given a DPS $\mathcal{D}$ and a scheduler $\mathcal{G}$, we construct the transition system $TS = (\mathcal{S}, y^{in}, \rightarrow)$ where $\mathcal{S} \subseteq \mathcal{Y}$ is a set of snapshots, with the initial snapshot given by $y^{in} = (\mathbf{s}^{in}, \emptyset, \emptyset) \in \mathcal{S}$.*

Given a snapshot $y = (\mathbf{s}, U, X) \in \mathcal{S}$, where $U = \{a_1, a_2, \ldots, a_m\}$ and $X = \{(a_1, e_1, t_1), \ldots, (a_m, e_m, t_m)\}$, we define the next snapshots from y as follows.

- *Let $\mathcal{G}(y)$ be the set of actions scheduled. Recall that $U \subseteq \mathcal{G}(y)$. Let $V = \mathcal{G}(y) \setminus U = \{b_1, b_2, \ldots, b_k\}$.*
- *For each action $b = (E_b, \pi_b) \in V$, we pick an event $e_b \in E_b$ according to π_b, with duration $\delta(e_b)$ and cost $\chi(e_b)$. This generates a list $Y = \{(b, e_b, \delta(e_b)) \mid b \in V, e_b \in E_b\}$. Note that all the events in $X \cup Y$ have pairwise disjoint locations.*
- *From $X \cup Y$, we pick the subset $E_{\min} = \{(a, e, t_{\min}) \mid t_{\min}$ is minimum across all triples in $X \cup Y\}$. We then update the snapshot as follows:*
  - *(i) For each $(a, e, t_{\min})$ in $E_{\min}$, set $\mathbf{s}_{loc(e)}$ to tgt_e, and remove a from $U \cup V$ and $(a, e, t_{\min})$ from $X \cup Y$.*
  - *(ii) For each (a, e, t) in $(X \cup Y) \setminus E_{\min}$, update t to $t - t_{\min}$, thus reducing the time to completion of e by $t_{\min}$.*

  This results in a new snapshot $(\mathbf{s}', U', X')$. Note that each probabilistic choice of events for the actions in V deterministically determines the next snapshot.

We label each transition from snapshot y to y' with a pair of sets $(E_V, E_{\min})$, where $V = \{b_1, b_2, \ldots, b_k\}$ is the new set of actions chosen by the scheduler at snapshot y, $E_V = \{e_{b_1}, e_{b_2}, \ldots, e_{b_k}\}$ is the set of events chosen corresponding to V and $E_{\min}$ is the set of triples corresponding to the events that complete their execution at y'. We write such a transition in the usual way as $y \xrightarrow{(E_V, E_{\min})} y'$. We associate a probability, time duration and cost with this transition as follows.

The probability associated with the transition is $p = \prod_{e_b \in E_V} \pi_b(e_b)$, where for action $b = (E_b, \pi_b)$ in V, the event e_b is probabilistically chosen from the set E_b via π_b. If $V = \emptyset$, $p = 1$. The time duration associated with the transition is the time $t_{\min}$ attached to each $(a, e, t_{\min}) \in E_{\min}$. The cost associated with the transition is the sum of $\chi(e)$, for all $(a, e, t_{\min}) \in E_{\min}$.

We claim that the probabilities we have attached to transitions transform the system into a Markov chain. Suppose $V = \{b_1, b_2, \ldots, b_k\}$ is the new set of actions chosen by the scheduler at a snapshot y. As observed earlier, each combination of events $\{e_1, e_2, \ldots, e_k\}$ generated by applying π_{b_i} to E_{b_i}, $i \in \{1, 2, \ldots, k\}$, results in a unique next snapshot. Hence the sum of the probabilities across all the successors of y adds up to 1. If $V = \emptyset$, there is only one outgoing transition with probability 1.

Statistical Analysis of DPS

Properties We are interested in checking linear time properties for distributed probabilistic systems. For traditional transition systems, these are combinations of safety and liveness properties. In a quantitative setting, we would typically like to make assertions about the total duration or the total cost of a run. Since our DPS model is not deterministic, we have to frame these questions in terms probabilities. For instance, we might ask if the probability of reaching a target state within time t is at least p.

A natural approach to checking such a property is to estimate the probability by a large number of simulations. For this, we need the additional constraint that the property can be checked within a bounded length run, so that we can effectively terminate each simulation with a yes or no verdict. Bounded duration properties can be formalized in notations such as bounded linear-time temporal logic (BLTL) [24]. In this paper we will not get into the details of BLTL but instead concentrate on reachability properties, defined informally.

The main shortcoming of naively estimating probabilities through random simulations is that we have no guarantee about the accuracy of the estimate. To perform this estimation in a principled manner, we need to frame the property of interest as a hypothesis testing problem.

Hypothesis testing We briefly overview the preliminaries of hypothesis testing before detailing the simulation procedure. For more details, see [14, 24].

Suppose we are given a DPS $(\mathcal{S}, \Sigma, \delta, \chi, A)$, a bounded reachability property described by a formula ϕ in a suitable notation, and a threshold γ. Our goal is to verify if ϕ is achieved with probability at least γ, which we write as $\Phi = Pr_{\geq\gamma}\phi$.

Let p be the probability of satisfying ϕ. To verify whether $p \geq \gamma$, we test the hypothesis $H : p \geq \gamma$ against its negation $K : p < \gamma$. Since a simulation-based test does not guarantee a correct result, there are two types of errors we encounter: (i) Type-I error: accepting K when H holds, and (ii) Type-II error: accepting H when K holds. We would like to ensure that the probabilities of Type-I and Type-II errors are bounded by pre-defined values (say) α and β, respectively.

Enforcing exact bounds on Type-I and Type-II errors simultaneously is hard, so we allow uncertainty using an indifference region [24]. We relax the test by providing a range $(\gamma-\delta, \gamma+\delta)$ for a given threshold δ. We now test the hypothesis $H_0 : p \geq \gamma + \delta$ against $H_1 : p \leq \gamma - \delta$. If the value of p is between $\gamma - \delta$ and $\gamma + \delta$, we say that the probability is sufficiently close to γ, so we are indifferent with respect to which of the hypothesis K or H are accepted.

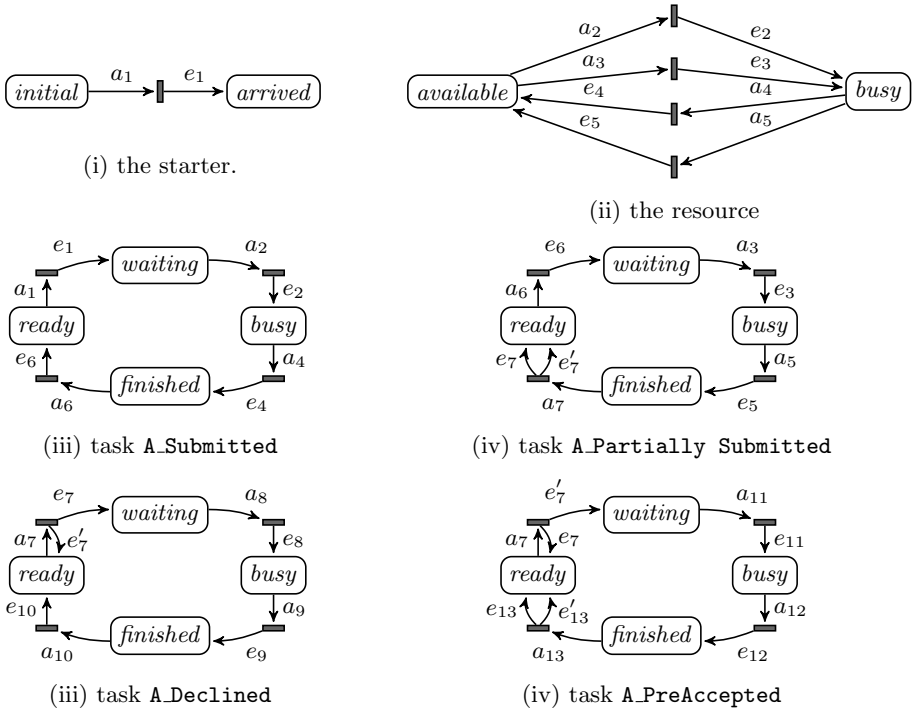
Sequential probability ratio test Traditional sampling theory fixes the sample size in advance based on the Type-I and Type-II error thresholds α and β. Computationally, it is often more efficient to estimate the sampling size adaptively, based on the observations made so far. Such an approach was proposed by Wald, called a sequential probability ratio test (SPRT) [23].

Time-bounded SPRT for DPS proceeds as follows. The user provides Type-I and Type-II error bounds α and β, as well the threshold of indifference δ. We

repeatedly simulate the system. We can determine in a bounded amount of time whether a simulation run satisfies the property of interest or not.

After m simulation runs, let d_m be the number of runs with a positive outcome so far. We calculate a ratio $quo = \frac{(\gamma^-)^{d_m}(1-\gamma^-)^{m-d_m}}{(\gamma^+)^{d_m}(1-\gamma^+)^{m-d_m}}$ that takes into account the number of successes and failures seen so far. We accept H_0 if $quo \leq \frac{\beta}{1-\alpha}$ and H_1 if $quo \geq \frac{1-\beta}{\alpha}$. Otherwise, we continue the simulation. The simulation is guaranteed to halt with probability 1 [24] and will typically converge much before the number of samples required by a traditional static estimate.

Please see Algorithm 1 in the Appendix for a concise algorithm.



(i) the starter.

(ii) the resource

(iii) task `A_Submitted`

(iv) task `A_Partially Submitted`

(iii) task `A_Declined`

(iv) task `A_PreAccepted`

Fig. 2. Modeling (part of) the running rcBPM system example as DPS agents.

4 Modeling rcBPM systems as DPS

We recall that an rcBPM system consists of a business process instantiated as a number of cases and a finite set of resources. Let us assume that in an rcBPM system $\mathcal{B}$, there are C cases numbered $\{1, 2, \ldots, C\}$ each with k_T tasks labelled as follows: T_{ij} denotes the j^{th} task of the i^{th} case, for $1 \leq i \leq C$ and $1 \leq j \leq k_T$. We assume the rate of the arrival process is λ cases per second. Let us assume there are r resources denoted $\{R_1, R_2, \ldots, R_r\}$. In the running example of loan/overdraft application, $k_T = 15$ and $r = 46$ (see Figure 1). For each resource R_i, let $tasks(R_i) \in \{T_{ij} \mid 1 \leq i \leq C, 1 \leq j \leq k_T\}$ denote the set of

tasks resource R_i is able to perform. Since resources can be shared among tasks, for any $1 \leq i, j \leq r$, $tasks(R_i) \cap tasks(R_j)$ can possibly be non-empty.

Given an rcBPM system, we transform it to a DPS as follows. We model tasks and resources as agents. To facilitate the arrival process and clearly mark the case completion, we also model the start and end states as agents. Hence, the rcBPM system $\mathcal{B}$ can be modeled using $r + C \times (k_T + 2)$ agents.

Each task agent consists of 4 states (i) *ready* to perform, (ii) *waiting* for a resource, (iii) *busy* being executed, and (iv) *finished*. Each resource agent consists of 2 states (i) *available* and (ii) *busy*. An agent modeling the start state is called a starter agent, and has two states *waiting* and *arrived*. Similarly, the agent modeling the end state is named finisher agent with states *pending* and *done*.

We illustrate a part of DPS modeling the running rcBPM example in Figure 2. We show the agents corresponding to the start state and 4 tasks: `A_Submitted`, `A_Partially Submitted`, `A_Declined` and `A_PreAccepted`. We also demonstrate a resource that can perform tasks `A_Submitted` and `A_Partially Submitted`. The states are depicted in rounded cornered rectangles and the edges between the states are defined as follows: $s \xrightarrow{\text{a} \quad \text{e}} s'$ denotes the action at local state s, $e \in E_a$ and s' be the next local state after event e.

The starter agent mimics the wait for a case. At state *initial*, it synchronizes with the starting task agent at *ready* state followed by an event with probability 1. The duration of the event is equal to the total time before arrival for the particular case. The starter then moves to *arrived* state and the starting task moves to *waiting* state, where it waits for a resource to be scheduled. For example, Figure 2(i) shows the action a_1 between the starter agent and the task `A_Submitted` followed by event e_1 such that $\pi_a(e_a) = 1$ and $\delta(e_1) = 1/\lambda$.

Once a task is in *waiting* state and the scheduler assigns a resource that is in *available* state, the task and the resource synchronize and they both move to *busy* state with probability 1. In Figure 2, examples of such actions are a_2 and a_3. There they synchronizes again, performs an event with possibly non-zero time duration, and the task moves to *finished* with probability 1. In Figure 2, examples of such actions are a_4 and a_5.

Once a task is finished, it signals the next task(s) in the control flow via synchronization. For example, in Figure 2, after task `A_Partially Submitted` is at *finished*, it sychronizes with tasks `A_Declined` and `A_Preaccepted` via action a_7 such that $E_{a_7} = \{e_7, e_7'\}$ with $\pi_{a_7}(e_7) = 0.84$ and $\pi_{a_7}(e_7') = 0.16$. Hence, with probability 0.84, event e_7 is chosen and only task `A_Declined` moves to *waiting* state. Otherwise, with probability 0.16, event e_7' is chosen and only task `A_Preaccepted` moves to *waiting* state. In both cases, `A_Partially Submitted` moves to *ready* state. A task is ready again after finishing since due to loops in control flow, a task may be executed multiple times in the same case.

When a case finishes, the last task in *finished* state synchronizes with the finisher agent in *pending* state. The task then moves to *ready* as usual and the finisher agent moves to *done*, indicating the completion of the corresponding task. The finisher agent then stays at the state *done* with probability 1.

Extensibility For brevity, we did not illustrate the rest of the DPS corresponding to the loan/draft application, but it can be easily extended. Also, we would like to point out that the described methodology to model rcBPM systems as DPS is only one example among many possibilities. One may easily extend this approach and incorporate an even more complex state space for tasks or resources modeling different scenarios. For example, we can easily model probabilistic error in task execution. Let us assume that when the resource depicted in Figure 2 performs task `A_Submitted`, there is a small probability of 0.1 that such an execution fails. This phenomenon can be easily modeled by adding another event e to E_{a_2} such that $\pi_{a_2}(e_2) = 0.9$ and $\pi_{a_2}(e) = 0.1$ where after e, they move back to *available* and *waiting* respectively. Also, for simplicity, we assumed that the business processes under consideration are sound, but DPS can be easily used to model unsound rcBPM systems as well.

5 Experimental Evaluation

We have tested our SMC procedure on the running example. The property we are interested in is follows:

With probability at least 0.99*, when cases arrive at a fixed rate of* 1 *case per* 10 *seconds, x fraction of cases are completed among C cases within t seconds.*

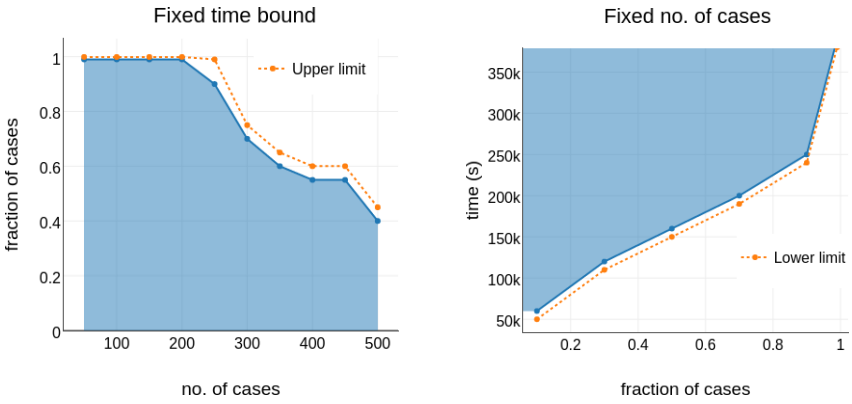


Fig. 3. (left) Fraction of cases completed vs total no. of cases when the time bound is fixed, (right) Minimum total time vs fraction of cases completed when the total number of cases is fixed.

We ensure that the probability of Type-I error and Type-II error of verifying this property is less than 0.01 with indifference region (0.99 ± 0.005).

We first investigate the relationship between the number of total cases and the number of cases completed within fixed maximum time $t = 500,000$ seconds. The shaded area in Figure 3 (left) represents the values (C, x) that satisfies the property when $t = 500,000$ seconds. The dotted line represents an upper limit for the values (C, x) satisfying the property.

Then, we illustrate the relationship between the minimum total time and the fraction of cases completed within that total time when the total number of cases $C = 100$ is fixed. The shaded area in Figure 3 (right) represents the values (x, t) that satisfies the property when $C = 100$. The dotted line represents a lower limit for the values (x, t) satisfying the property. We note that the limits are not the tightest, but can be made arbitrary closer with more SMC simulations.

Extensibility Though our experiment is only a proof-of-concept, we would like point out that we can scale the size of the model comfortably to accommodate 500 cases. Since there is no other known approach to verify business processes with provable bounds, we were not able to compare our results with existing literature. This methodology also supports verifying a variety of properties, depending on the focus of optimization for the business. For example, one may investigate the performance of resources by verifying the following property: given a resource, in what fraction of cases was it used?

6 Conclusion

We have presented a modular approach to modelling resource-constrained BPM systems with multiple cases, using distributed probabilistic systems. We have shown that a real-time distributed probabilistic system under a fixed scheduler behaves like a Markov chain. We have then presented a rigorous technique for time-bounded approximate verification of business processes using statistical model checking, illustrated through a proof-of-concept experiment.

In future, we plan to extend this model to shed light on different types of scheduling policies and their impact on business processes. We would also like to incorporate stochastic durations for events, which will take us to a Continuous Time Markov Chain (CTMC) setting. Finally, we would like to see how approximate verification techniques can also enrich process mining BPM systems [2].

Acknowledgements The authors would like to thank S Akshay for his invaluable comments on the draft and Ansuman Banerjee for the early discussions.

References

1. W. M. P. Aalst. Verification of workflow nets. In Pierre Azma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, number 1248 in LNCS, pages 407–426. Springer Berlin Heidelberg.
2. W. M. P. Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011.
3. W. M. P. Aalst. Business process management: a comprehensive survey. ISRN Softw. Eng. 1-37. *ISRN Software Engineering*, (1), 2013.
4. W. M. P. Aalst. Business process management as the Killer App for Petri nets. *Software & Systems Modeling*, 14(2):685–691, June 2014.
5. W. M. P. Aalst. *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, chapter Business Process Simulation Survival Guide, pages 337–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

6. W. M. P. Aalst and K. M. V. Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2004.
7. R. P. J. C. Bose and W. M. P. Aalst. *Business Process Management Workshops: BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, pages 221–222. Springer Berlin Heidelberg, 2013.
8. K. R. Braghetto, J. E. Ferreira, and J.-M. Vincent. Performance evaluation of resource-aware business processes using stochastic automata networks. *Int. Journal of Innovative Computing, Information and Control*, 8(7B):5295–5316, 2012.
9. L. I. N. Chuang, Q. U. Yang, R. E. N. Fengyuan, and Dan C. Marinescu. Performance Equivalent Analysis of Workflow Systems Based on Stochastic Petri Net Models. In *Engineering and Deployment of Cooperative Information Systems*, number 2480 in LNCS, pages 64–79. Springer Berlin Heidelberg, 2002.
10. M. Dumas, M. L. Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer Berlin, Berlin, Heidelberg.
11. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games and distributed control for asynchronous systems. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, Proceedings*, pages 455–465, 2004.
12. L. T. Herbert. *Specification, Verification and Optimisation of Business Processes*. A Unified Framework. Technical University of Denmark, 2014.
13. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, number 3920 in LNCS, pages 441–444. Springer Berlin Heidelberg, January 2006.
14. A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *Runtime Verification*, pages 122–135. Springer, 2010.
15. M. Magnani and D. Montesi. BPMN: How Much Does It Cost? An Incremental Approach. In *Business Process Management*, number 4714 in LNCS, pages 80–87. Springer Berlin, September 2007.
16. M. Netjes, W. M. P. Aalst, and Hajo A R. Analysis of resource-constrained processes with colored petri nets. In *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 576, pages 251–266, 2005.
17. C.A.L. Oliveira, R.M.F. Lima, H.A. R., and J.T.S. Ribeiro. Quantitative analysis of resource-constrained business processes. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(3):669–684, May 2012.
18. H. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Springer, July 2003.
19. N. Russell, A. H. M. Ter Hofstede, and N. Mulyar. Workflow ControlFlow Patterns: A Revised View. Technical report, 2006.
20. R. Saha, J. Esparza, S. K. Jha, M. Mukund, and P. S. Thiagarajan. Distributed Markov chains. In *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, 2015. Proceedings*, pages 117–134, 2015.
21. P. Sampath and M. Wirsing. Computing the Cost of Business Processes. In *Information Systems: Modeling, Development, and Integration*, number 20 in LNCS, pages 178–183. Springer, April 2009.
22. B.F. van Dongen. BPI challenge 2012, 2012.
23. A. Wald. Sequential tests of statistical hypotheses. *Ann. Math. Statist.*, pages 117–186, 1945.
24. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 2004.

A Appendix

Algorithm 1 Statistical Model Checking for DPS

INPUT:

$\mathcal{D}, \mathcal{G}$ ▷ a DPS and a scheduler

$\Phi = Pr_{\geq\gamma} T_{\leq t} \phi$ ▷ a property

α, β, $\delta \in (0,1)$ ▷ error bounds and threshold of indifference

OUTPUT:

YES or NO

procedure SIMULATE-DPS

  $m \leftarrow 0$ ▷ the number of simulations so far

  $\gamma^+ \leftarrow \gamma + \delta$ and $\gamma^- \leftarrow \gamma - \delta$

  while True **do**

    $t_{spent} \leftarrow 0$ ▷ time spent so far

    $y^{in} = (\mathbf{s}^{in}, \emptyset, \emptyset), y = (\mathbf{s}, U, X) = y^{in}$ ▷ the initial and current snapshot

    $\rho \leftarrow y$ ▷ the current execution

    $b \leftarrow 0$ ▷ the outcome of the Bernoulli random variable

    $d_m \leftarrow 0$ ▷ accumulator of outcome of the Bernoulli variable

    while $(t_{spent} \leq t)$ **do**

      $\mathcal{G}(y) = U \cup V \leftarrow$ *scheduled actions at* y

      $Y \leftarrow$ *set of fresh actions probabilistically chosen from* V

      $E_{\min} \subseteq X \cup Y$ *is the set of tuples with minimum time to completion*

      for all $(a, e, t_{\min}) \in E_{\min}$ **do**

        $\mathbf{s}_{loc(e)} \leftarrow tgt_e$, *remove* a *from* $U \cup V$, *remove* (a, e, t) *from* $X \cup Y$

      for all $(a, e, t) \in X \cup Y \setminus E_{min}$ **do**

        $t \leftarrow t - t_{\min}$

      $y = (\mathbf{s}', U', X')$ *is the new snapshot*

      $t_{spent} = t_{spent} + t_{\min}, \rho \leftarrow \rho y$

      if ρ satisfies ϕ **then**

        $b = 1$

        break

    $m \leftarrow m + 1$ and $d_m \leftarrow d_m + b$

    $quo = \frac{(\gamma^-)^{d_m}(1-\gamma^-)^{m-d_m}}{(\gamma^+)^{d_m}(1-\gamma^+)^{m-d_m}}$

    if $(quo \geq \frac{(1-\beta)}{\alpha})$ **then**

      return NO

    else if $(quo \leq \frac{\beta}{1-\alpha})$ **then**

      return YES

Table 1. Resource-task matrix (truncated) with average time taken by the resource.

	10228	**10629**	**10779**	**10859**	**10861**	**10862**	**10863**	**10880**	**10881**	**10889**	**10899**	**10909**	**10910**	**10912**	**10913**	**10929**	**10931**
A_Accepted	2521			30435	47184	53086	94678	62187	7210	18276		24136	38871	62747	71002	30092	7475
A_Approved																	
A_Cancelled			51106														
A_Declined																	
A_Finalized	0			35	62	0	59	78		86		24	26	0	79	94	25
A_PartlySubmitted																	
A_Preaccepted	31897	2186		5775	9823	1061	3333	17990	44545	21613		16301	18480	20330	17735	19720	29301
A_Submitted																	
O_Cancelled				173747	140840	32		37	8142	53509		19437	155656		0	0	0
O_Created	47			1	1	2	1	2	2	2		1	2	1	9	1	1
O_Selected	130			0	48	38587	179	83	0	27032		42117	28	192	18598	24747	22901
O_Sent	0			0	0	0	0	0	0	0		0	0	0	0	0	0
O_Sent_Back											85338						
Dummy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O_Accepted	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0