Fluid Rewards for a Stochastic Process Algebra

Mirco Tribastone, Jie Ding, Stephen Gilmore, and Jane Hillston

Abstract—Reasoning about the performance of models of software systems typically entails the derivation of metrics such as throughput, utilisation, and response time. If the model is a Markov chain, these are expressed as real functions of the chain, called *reward models*. The computational complexity of reward-based metrics is of the same order as the solution of the Markov chain, making the analysis infeasible when evaluating large-scale systems. In the context of the stochastic process algebra PEPA, the underlying continuous-time Markov chain has been shown to admit a deterministic (fluid) approximation as a solution of an ordinary differential equation, which effectively circumvents state-space explosion. This paper is concerned with approximating Markovian reward models for PEPA with *fluid rewards*, i.e., functions of the solution of the differential equation problem. It shows that (a) the Markovian reward models for typical metrics of performance enjoy asymptotic convergence to their fluid analogues, and that (b), via numerical tests, the approximation yields satisfactory accuracy in practice.

Keywords—Modeling and Prediction, Ordinary Differential Equations, Markov processes.

1 Introduction

Stochastic process algebras have emerged in the last twenty years as a valuable modelling paradigm in software performance engineering [1]. For example, a strong link has been established between model-driven development based on the UML and assessment of non-functional properties using stochastic process algebra languages such as PEPA [2], [3]. However, like all discrete state system description techniques, stochastic process algebras suffer from the problem of *state space explosion*, which can make analysis extremely costly or even infeasible. This problem is exacerbated when, as is often the case with software models, we wish to evaluate alternative configurations of a systems in order to choose between different degrees of replication and redundancy whilst ensuring that the system is dimensioned adequately to meet user requirements. These problems have motivated recent work on the approximation of the continuous-time Markov chain (CTMC) underlying a stochastic process algebra model with ordinary differential equations (ODEs) [4].

In a typical setting considered in this paper, the state representation is a vector of nonnegative integers in which each element denotes the population count of a specific kind of entity of the system under study, a distinct component type in the corresponding stochastic process algebra model. The semantics is generally given in terms of a CTMC, and each configuration of the system will be mapped to a CTMC $\{X(t)\}$. The following programme is carried out to determine an approximating differential equation. Given a Markovian model of the system under scrutiny, we must find some system parameter n and build a sequence of CTMCs, denoted by $X_n(t)$, where each CTMC represents an instance of the system with a specific value of n. At an intuitive level n can be thought of as the *scale* of the current system configuration.

Under conditions related to how the transition rates behave as a function of n, it holds that a sample path of the normalised real-valued stochastic process $X_n(t)/n$ is asymptotically indistinguishable from $x(t)$, the solution of an initial value problem associated with a system of coupled ordinary differential equations. This result justifies the approximation $X_n(t) \approx nx(t)$, which is crucial from a computational standpoint because it allows us to estimate a difficult—if not intractable—$X_n(t)$ with a much more pleasant $x(t)$.

The main contribution of [5] was to adopt this differential-analysis framework for the purpose of evaluating the performance of software systems described with PEPA [6]. Each element of the state descriptor counts the number of copies of the components which exhibit a particular state. In some models knowing the trajectory of the population counts over time is sufficient to gain deep insight into the system under scrutiny. However, in situations concerned with performance modelling, metrics of interest are usually described by means of quantities derived from the Markov process.

This approach is known in the literature as the *Markov reward model*. Specifically, this paper will be concerned with *rate rewards*, i.e. functions which associate a real value with each state of the CTMC. Let $\mathcal{X}$ be the state space of a CTMC $X(t)$, a rate reward is therefore formally represented by a function $\rho : \mathcal{X} \to \mathbb{R}$. The stochastic process $\rho(X(t))$ is called a *reward model*. Reward models have been widely employed in *performa-*

- *M. Tribastone is with the Insitut für Informatik, Ludwig-Maximilians-Universität München, Germany. E-mail: tribastone@pst.ifi.lmu.de*
- *J. Ding is with the College of Information Engineering of Yangzhou University, China. E-mail: jieding@yzu.edu.cn*
- *S. Gilmore and J. Hillston are with the Laboratory for Foundations of Computer Science, School of Informatics, Edinburgh University, UK. E-mail: {stg,jeh}@inf.ed.ac.uk*

bility analysis, which is concerned with the composite evaluation of performance and reliability measures of degradable computer systems [7]. Performability metrics may be defined through ρ. The *accumulated reward* $Y(t)$ is a transient measure which gives the area under $\rho(X(t))$, i.e., $Y(t) \stackrel{def}{=} \int_0^t \rho(X(s))ds$ and the *time-averaged* reward $W(t)$ divides the accumulated reward over the length of the time period, i.e., $W(t) \stackrel{def}{=} Y(t)/t$. For example, the most basic form of *availability* may consist of a reward structure Av which assigns the reward 1.0 to each operational state of the chain and 0 to the non-operational states (e.g., [8], [9]). Thus, $\mathbb{E}[Av(t)]$ gives the average instantaneous availability of the system at time t and the total availability over the interval $[0, t]$ is given by $\mathbb{E}\left[\int_0^t \rho(Av(s))ds\right]$. Considerable attention has been paid to the evaluation of the cumulative distribution of $Y(t)$—an extensive review of solution techniques is provided in [10] (in particular Section 3.3).

Clearly, the framework of Markov reward models may be used for the evaluation of purely performance-related measures. This appeared as early as in 1978 in the work of Beaudry where the notion of *computation availability* is related to the expected value of a reward structure called *computation capacity*, which gives the amount of processing power of a system at any point in time [11]. Trivedi *et al.* give a taxonomy of performance evaluation reward models in [9], which includes examples of throughput [7], bandwidth specification [8], and average response time [12].

1.1 Paper Contributions

When the CTMC is inferred from a model specification language, it is of utmost importance to be able to define the Markov reward model directly in terms of the constituents of the high-level description (e.g., [13], [14] and, more recently, [15]). In this respect, the first contribution of this paper is to define the notions of *action throughput*, *capacity utilisation*, and *average response time* as reward structures which may be transparently inferred from the process algebraic description through the semantics presented in [5]. Throughput measures the frequency of execution of activities with given PEPA action types and is analogous to the notion of throughput in stochastic Petri nets and in queueing networks. Throughput-like measures may also include behaviours with different meanings, such as effective bandwidth and loss rate in communication networks. Finally, the definition of throughput is used to derive average response times for the completion of a set of activities by a PEPA component. Capacity utilisation gives a measure of the likelihood that a sequential component is idle due to the unavailability of another synchronising component in the system.

In general, from a computational standpoint, the evaluation of these reward models is at least as difficult as solving the CTMC for a transient or the equilibrium distribution. This is because the probability vector has to be computed first, and successively the reward measure may be obtained by multiplying the reward in each state by its probability, summing across all the states. Therefore, this approach becomes infeasible for large-scale models due to the problem of state-space explosion. The other main contribution of this paper is to build upon the deterministic approximation $X_n(t) \approx n\,x(t)$ and characterise under which conditions the similar approximation $\rho\big(X_n(t)\big) \approx \rho\big(x(t)\big)\rho'(n)$, where ρ' is a reward-dependent deterministic function, holds for a reward model. This relationship has a crucial implication because it permits estimations of performance indices at a dramatically reduced computational cost—given the ODE solution $x(t)$, the deterministic approximation of a reward model only entails a single evaluation of the real function ρ, for any time point t of interest.

As with the underlying CTMC of PEPA, the indices of performance examined in this paper are shown to enjoy convergence to their deterministic estimates asymptotically as $n \to \infty$. Since this relation cannot be used for the quantitative assessment of the accuracy of the approximation, the convergence is studied by means of numerical tests on many randomly-generated PEPA models. Measures of throughput, utilisation, and average response times for such models are evaluated both deterministically and stochastically through simulation. This investigation gives confidence that the deterministic evaluation behaves satisfactorily at low population levels and shows good rate of convergence with increasing problem sizes.

1.2 Related Work

The question of defining performance measures in terms of the PEPA components has been studied since its inception [6] and has subsequently been considered also in logical terms [16]. The extraction of fluid performance measures from PEPA is discussed in [17], where the authors introduce a slight variant to the language in order to be able to express time-to-absorption measures. The comparison against the corresponding stochastic analysis is only empirical and does not make use of properties of convergence between the two interpretations. By contrast, the throughput and average response-time calculations presented in [18] and [19] are generalised in the present framework and can be shown to be embodied in it.

1.3 Paper Organisation

The technical contributions presented in this paper are taken from [20] (especially, Chapter 5). In order to provide a self-contained report, Section 2 gives an informal overview of PEPA and its differential analysis. The main theoretical results of convergence of the performance rewards are presented in Section 3. Throughput, capacity utilisation and average response time are motivated and formally defined in Sections 4, 5 and 6, respectively. Throughout the paper, a simple running example will

be used to illustrate these notions and provide some preliminary results on the quality of the differential approximation. Then, Section 7 presents the results of numerical validation on a more computationally challenging model with faster rate of state-space growth. Finally, Section 8 gives concluding remarks, commenting on the computational advantages of fluid rewards.

2 Overview of PEPA

This section presents background material concerning PEPA and its semantics for Markovian and deterministic analysis. The concepts are introduced by means of a running example, which will be used in the remainder of the paper to apply the definitions of the performance indices to a concrete case study. The interested reader should consult [6] for the formal development of the language and its Markovian interpretation, and [5] for the deterministic semantics.

A system in PEPA is described as a composition of *sequential components*, i.e., automata evolving through a number of local states (*derivatives*) and synchronising over shared action types. Consider the following basic model of client/server interaction in PEPA:

$$\begin{aligned} Download &\stackrel{def}{=} (transfer, r_1).Think \\ Think &\stackrel{def}{=} (think, r_2).Download \\ Upload &\stackrel{def}{=} (transfer, r_3).Log \\ Log &\stackrel{def}{=} (log, r_4).Upload \\ System &\stackrel{def}{=} Download[N_C] \underset{\{transfer\}}{\bowtie} Upload[N_S] \end{aligned} \tag{1}$$

The first two definitions describe the cyclic behaviour of a client, which performs a $transfer$ action and pauses before engaging in the activity again. The action $transfer$ is performed in synchronisation with a sequential component of a server, when it exhibits the derivative $Upload$. The client's state $Think$ carries out an *independent action*, i.e., an action which is not synchronised with any other sequential component in the system. Similarly, Log executes an independent action by the server component.

Each transition has a rate of execution r, which identifies an exponentially distributed duration of the action with mean $1/r$ time units. Each sequential component has a local view of the rate of the shared action, and the semantics of PEPA defines the rate of the synchronising activity as the minimum of the rates involved. In this case, the rate of the shared action between one client and one server is $\min(r_1, r_3)$.

Given a sequential component S, the *array notation* $S[N]$ denotes N copies of identically behaving components. Thus, the equation $System$ says that there is a pool of N_C copies of a client and a pool of N_S copies of servers, and that each pair cooperates over the action types in the set L, as indicated by the operator $\underset{L}{\bowtie}$.

Another important operator of the language, *choice*, denoted by $+$, enables all the activities of its operands, and the outcome is treated stochastically. For instance, $(\alpha, r).P + (\alpha, s).Q$ describes a process which will behave as P (resp., Q) with probability $\frac{r}{r+s}$ (resp., $\frac{s}{r+s}$). A model with choice will be presented in Section 7.1.

When interpreted against the Markovian semantics, the model in (1) gives rise to a finite CTMC. Each state of this chain is a derivative of $System$, which records the local derivatives exhibited by each sequential component. For instance, if $N_C = N_S = 1$, the underlying CTMC is characterised by the following states:

- $Download \underset{\{transfer\}}{\bowtie} Upload$, which denotes that the client and the server can perform the shared $transfer$ action.
- $Think \underset{\{transfer\}}{\bowtie} Log$, in which both sequential components may perform their independent action.
- $Think \underset{\{transfer\}}{\bowtie} Upload$, in which only the client component performs its independent action.
- $Download \underset{\{transfer\}}{\bowtie} Log$, in which only the server component performs its independent action.

The differential interpretation is based on the *reduced context* of a PEPA model, denoted by $red(\cdot)$, which abstracts away from the actual population levels described in the system equation and only collects the information on all the local states exhibited by the sequential components and the structure of the cooperation between them. The reduced context of $System$ in (1) is:

$$red(System) = Download \underset{\{transfer\}}{\bowtie} Upload$$

This representation is sufficient to define a population vector ξ, the state descriptor of the PEPA model (*numerical vector form*), of length hereafter denoted by d. In general, let $\mathcal{C}_i$ be the derivative set of the i-th component, $i = 1, 2, \ldots, N_C$ and let N_i be its size, i.e., $N_i = |\mathcal{C}_i|$. Let $C_{i,j}$ denote the j-th derivative of the i-th component, $j = 1, 2, \ldots, N_i$. The state descriptor ξ assigns a coordinate, denoted by $\xi_{i,j}$, to each local derivative $C_{i,j}$ and indicates the number of copies in the system which exhibit that derivative. The following mapping is assumed for the model (1): $C_{1,1} = Download, C_{1,2} = Think, C_{2,1} = Upload, C_{2,2} = Log$. (Sometimes, the following one-dimensional indexing of ξ may also be used: $Download \mapsto \xi_1$, $Think \mapsto \xi_2$, $Upload \mapsto \xi_3$, $Log \mapsto \xi_4$.)

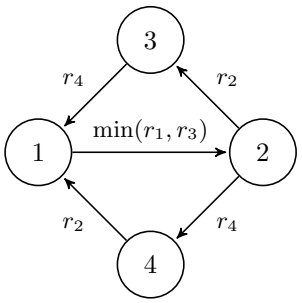
The deterministic semantics gives rise to Lipschitz continuous *generating functions* of the underlying *population-based* CTMC. For each state ξ of the CTMC, each of these functions, denoted by $\varphi_\alpha(\xi, l)$, gives the rate at which it jumps to state $\xi + l$ and the corresponding action type α which must be performed to make this jump. The *jump vector* l records the impact of the given action on the population vector. In (1), the generating functions are defined as follows:

$$\varphi_{transfer}\big(\xi, (-1, 1, -1, 1)\big) = \min(r_1\xi_1, r_3\xi_3) \tag{2}$$

$$\varphi_{think}\big(\xi, (1, -1, 0, 0)\big) = r_2\xi_2 \tag{3}$$

$$\varphi_{log}\big(\xi, (0, 0, 1, -1)\big) = r_4\xi_4 \tag{4}$$

and $\varphi_\alpha(\xi, l) = 0$ for all other action types α and $l \in \mathbb{Z}^d$. For instance, (2) states that, in a system with ξ_1 clients and ξ_3 servers, the rate of completion of the action



State	*Client*		*Server*	
	D	*T*	*U*	*L*
1	1	0	1	0
2	0	1	0	1
3	1	0	0	1
4	0	1	1	0

Fig. 1: State space in the numerical vector form of (1) with $N_C = N_S = 1$. D, T, U, and L stand for *Download*, *Think*, *Upload*, *Log*, respectively.

transfer is $\min(r_1\xi_1, r_3\xi_3)$. This causes a transition to a state in which the number of *Download* components is decreased by one and, correspondingly, the number of *Think* components is increased by one. Since it is a shared action, a similar behaviour is observed for the servers, i.e., the number of *Upload* components is decreased by one and the number of *Log* components is increased by one. Figure 1 shows the population-based CTMC for the simple case $N_C = N_S = 1$.[1]

The explicit enumeration of the state space is not required for the generation of the differential equation. Rather, this is directly derived from the generating functions as

$$\frac{dx(t)}{dt} = F\big(x(t)\big) = \sum_{l\in\mathbb{Z}^d} l \sum_{\alpha\in\mathcal{A}} \varphi_\alpha\big(x(t), l\big),$$

where $\mathcal{A}$ is the set of all action types defined in the model ($\mathcal{A} = \{transfer, think, log\}$ in (1)). The differential equation underlying (1) is

$$\begin{aligned}
\frac{dx_1(t)}{dt} &= -\min\left(r_1x_1, r_3x_3\right) + r_2x_2 \\
\frac{dx_2(t)}{dt} &= \min\left(r_1x_1, r_3x_3\right) - r_2x_2 \\
\frac{dx_3(t)}{dt} &= -\min\left(r_1x_1, r_3x_3\right) + r_4x_4 \\
\frac{dx_4(t)}{dt} &= \min\left(r_1x_1, r_3x_3\right) - r_4x_4
\end{aligned} \tag{5}$$

By inspecting *System* in (1) it is possible to extract the initial state of the system, $\delta = (N_C, 0, N_S, 0)$. The (unique) solution to the initial-value problem $dx(t)/dt = F\big(x(t)\big)$, with $x(0) = \delta$, gives the deterministic time-course trajectory $x(t)$ of the population levels of all sequential components. It is possible to show that the solution must always satisfy a form of *conservation law*. In this example, observe that $dx_1(t)/dt + dx_2(t)/dt = 0$ and $dx_3(t)/dt + dx_4(t)/dt = 0$. Given the initial condition δ, this implies that $x_1(t)+x_2(t) = N_C$ and $x_3(t)+x_4(t) = N_S$ for all t, which states that no components are created or destroyed during the temporal evolution of the process. Furthermore, at any point in time there must be at least one strictly positive component population level.

1. In this case, the size of the state space is equal to that derived from the original semantics. However, if the initial population levels are increased then the state space based on the population-vector form is smaller because the semantics exploits equivalence relations on the process algebra terms.

TABLE 1: Summary of notation.

Symbol	*Description*
$\mathcal{A}$	Set of all action types in a PEPA model
$C_{i,j}$	Local derivative in the reduced context
	Index i ranges across distinct sequential components
	Index j ranges across derivatives of the i-th component
$\xi \in \mathbb{N}^d$	Population vector (elements $\xi_{i,j}$)
$l \in \mathbb{Z}^d$	Jump vector of a CTMC transition (elements $l_{i,j}$)
$\delta \in \mathbb{N}^d$	Parameter for the initial state of the CTMCs
$X_n(t)$	Population-based CTMC with initial state $X_n(0) = n\delta$
$\varphi_\alpha(\xi, l)$	Generating function of a CTMC, $\alpha \in \mathcal{A}$
$x(t)$	Fluid limit of the CTMC ($t = \infty$ at equilibrium)
$\rho(\omega)$	Reward function ρ with generic argument ω, written
	$\rho\left(X_n(t)/n\right)$ for a Markovian reward
	or $\rho\left(x(t)\right)$ for a fluid reward
$\xrightarrow{\mathbb{P}}$	Convergence in probability (cf. Eq. 7)
$\mathbb{E}\left[Y\right]$	Expectation of the random variable Y

The result of convergence presented in [5] relates $x(t)$ to a family of CTMCs $\{X_n(t)\}$ such that the initial state of the n-th CTMC has n times more components than the chain $X_1(t)$. Such a family, denoted by $\{X_n(t)\}$ is parametrised by an integer n, called the *scale factor*, by letting $X_n(0) = n\delta$. This corresponds to increasingly large populations as functions of n, though the relative proportions between the initial populations of the sequential components are constant across the family of CTMCs, i.e., the ratio between clients and servers is always N_C/N_S.

The normalised process (also referred to as the *density process*) $X_n(t)/n$ plays a crucial role in the theory of convergence of PEPA models, because the sequence $\{X_n(t)/n\}$ is shown to converge to the deterministic limit in the sense that

$$\lim_{n\to\infty} \mathbb{P}\left(\sup_{s\le t} \left|X_n(s)/n - x(s)\right| > \varepsilon\right) = 0, \quad \forall\varepsilon > 0. \tag{6}$$

Intuitively, this result establishes that for sufficiently large n, the solution to the differential equation is as good as a sample path of the normalised process over any finite time interval.

Notation: For ease of reference, Table 1 summarises the notation used throughout this paper.

3 Fluid Approximation of Rewards

To illustrate that the ODE solution $x(t)$ is not always sufficient to gain insight into the performance characteristics of models of computer systems, consider, for instance, two configurations of (1), in which all rates of one instance (i.e., r_1, r_2, r_3, r_4) are doubled with respect to the rates of the other instance. The solutions to the

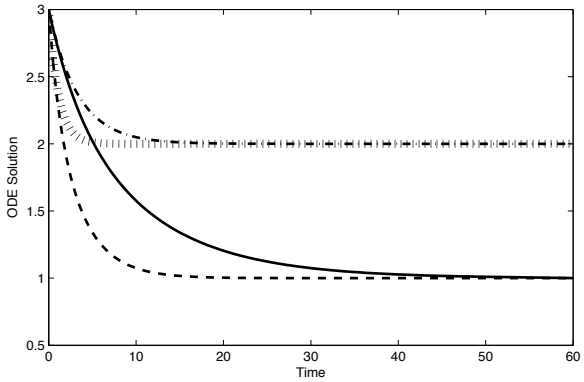


Fig. 2: Deterministic trajectories of $x_1(t)$ (i.e., *Download*) and $x_3(t)$ (i.e., *Upload*) of Eq. 1 for two distinct configurations which have the same initial conditions but the rates in the *fast* model are obtained by doubling the rates in the *slow* model. Solid line: slow $x_1(t)$, dash-dotted line: slow $x_3(t)$, dashed line: fast $x_1(t)$, dotted line: fast $x_3(t)$.

underlying ODEs (5), with the same initial condition, is depicted in Fig. 2 for $x_1(t)$ and $x_3(t)$. It reveals similar trajectories after approximately fifty time units. Indeed, it is possible to show that any pair of model instances such that the rates of one instance are multiples (with the same factor) of the rates of the other instance have the same equilibrium distribution of the underlying CTMC (hence, of the normalised process $X_n(t)/n$, for any n). However, this fails to capture the basic intuition that one model should be *faster* than the other, because of the rate configurations used. As will be shown in Section 4, the different behaviours of these two models are captured by the reward structure for the calculation of throughput.

3.1 A Generic Framework

The remainder of this section is concerned with a general set-up of the framework within which will be defined all the reward structures presented in this paper. The convergence property (6) implies that, for any fixed t, the sequence of random variables $\{X_n(t)/n\}$ converges *in probability* toward $x(t)$ (as observed, e.g., in [21]):

$$\lim_{n\to\infty} \mathbb{P}\big(|X_n(t)/n - x(t)| > \varepsilon\big) = 0, \quad \text{for every } \varepsilon > 0. \quad (7)$$

From now on, this form of convergence will be denoted by the usual notation $\xrightarrow{\mathbb{P}}$, e.g., $X_n(t)/n \xrightarrow{\mathbb{P}} x(t)$. The main objective of this section is to determine under which conditions convergence in probability of the density process implies convergence for the reward model in the form $\rho\,(X_n(t)/n) \xrightarrow{\mathbb{P}} \rho(x(t))$. This constitutes the formal justification of the use of the deterministic approximation for the computation of performance metrics from PEPA models. The reasoning will be mostly based upon the Continuous Mapping theorem, which ensures convergence in probability for functions of stochastic variables.

Theorem 1 (Continuous Mapping (cf. [22], Section 29))**.** *Let Y_n be a random variable with ranges in $\mathbb{R}^d$ and $Y_n \xrightarrow{\mathbb{P}} c$, with $c \in \mathbb{R}^k$. Let $g : \mathbb{R}^d \to \mathbb{R}^k$ be continuous at c. Then,*

$$g(Y_n) \xrightarrow{\mathbb{P}} g(c).$$

This result is directly applicable to study the convergence of $\rho\,(X_n(t)/n)$ toward $\rho(x(t))$ by letting $Y_n(t) = X_n(t)/n$, for any t. In general, however, the performance index of interest for a CTMC of a PEPA model is expressed as a reward $\rho\,(X_n(t))$. Therefore, metric specifications will be restricted to reward structures which are not explicitly dependent upon the scaling factor n. In other words, the reward structure ρ must satisfy the condition that there exists some ρ' such that

$$\rho(X_n(t)/n) = \rho(X_n(t))/\rho'(n). \quad (8)$$

Then, the asymptotic convergence in probability $\rho\,(X_n(t)/n) \xrightarrow{\mathbb{P}} \rho(x(t))$ intuitively means that, for sufficiently large n,

$$\rho\,(X_n(t)) \approx \rho'(n)\rho(x(t)), \quad (9)$$

which gives an approximate estimate of $\rho\,(X_n(t))$ in terms of the deterministic quantity $\rho(x(t))$, as required. The performance metrics defined in this paper are developed within this framework. Specifically, they will be expressed in terms of the generating functions, i.e., $\rho(\varphi_\alpha(\omega, l))$, hence the verification of these conditions can be derived from the properties of φ. This is particularly useful because φ has been proven to be continuous and to give rise to a family of density-dependent CTMCs. Specifically, it implies that the condition in (8) is met, since in [5] it has been shown that

$$\varphi_\alpha(X_n(t)/n, l) = \varphi_\alpha(X_n(t), l)/n, \quad \forall\, l \in \mathbb{Z}^d, \alpha \in \mathcal{A}. \quad (10)$$

The cases of throughput, capacity utilisation, and response time will be dealt with more in detail in this paper because of the central role that the play in the specification of performance measures with many modelling techniques. However, the framework put forward here is more general and also includes other interesting indices. For instance, if ρ is a reward function satisfying the conditions presented above, then its definite integral is also a reward with guaranteed convergence to its deterministic limit. As a result, accumulated and time-averaged rewards (cf. Section 1) based on ρ are indices with meaningful fluid approximations. Other functions of suitable reward indices can be shown to enjoy convergence, such as linear combinations $A\,\rho_1(\cdot) + B\,\rho_2(\cdot) + \ldots$ with A and B constants, or minimums between rewards, e.g., $\min(\rho_1(\cdot), \rho_2(\cdot))$.

However, there are measures which do not exhibit convergence in the sense described in this paper. For example, consider a scenario with breakdowns and repairs, and let i be the index in the population vector that denotes the number of broken elements. A reward $R(\omega)$ may define two levels of availability, 1.0 and 0., by

setting $R(\omega) = 1.0$ if $0 \leq \omega_i < k$ and $R(\omega) = 0.0$ if $\omega_i \geq k$, for some $k > 0$. Clearly R is discontinuous at k.

In addition to such rewards with discrete values, problems also arise with rewards which depend on second-order moments of the CTMC [23] and functions of a reward which do not satisfy the conditions for convergence. In particular (8) does not imply that in general $\exp\{\rho(X_n(t)/n)\}$ can be written as $\rho' \exp\{\rho(X_n(t))\}$.

3.2 Accuracy of the Approximation

Although the results of asymptotic convergence are important from a theoretical standpoint for the justification of the use of the differential reward evaluation, they do not provide estimates of the approximation error for finite scale factors. The problem of assessing the accuracy quantitatively is clearly of great significance in most applications. In particular, it is often important to establish the accuracy for small scale factors because even for such factors the associated CTMC may be too large to permit feasible solution (either numerically or via stochastic simulation, cf. Section 8.1) and thus deterministic analysis constitutes the most convenient form of evaluation available. Unfortunately, theoretical bounds developed in the context of density dependent Markov chains (e.g., [24]) cannot be used here because in general $\rho(X_n(t))$ does not enjoy the Markov property (cf., e.g., [25], [26]). Here, similarly to [5], the accuracy for finite scale factors will be gauged more pragmatically by making a direct comparison between the expectation of the Markovian reward and its corresponding deterministic evaluation, using the following notion of percentage relative error:

$$\begin{aligned} \text{Error } \% &= \left| \frac{\mathbb{E}\left[\rho(X_n(t)/n)\right] - \rho(x(t))}{\mathbb{E}\left[\rho(X_n(t)/n)\right]} \right| \times 100 \\ &= \left| \frac{\mathbb{E}\left[\rho(X_n(t))\right] - \rho'(n)\rho(x(t))}{\mathbb{E}\left[\rho(X_n(t))\right]} \right| \times 100. \end{aligned} \tag{11}$$

3.3 Steady-State Rewards

Strictly speaking, the framework presented here could not be used asymptotically as $t \to \infty$ because the result of convergence (6) holds for finite t. Nevertheless, with a slight abuse, in the remainder of the paper steady-state rewards will be considered (for instance, in Section 6). In all subsequent discussions, these are interpreted as being given by approximating rewards computed at a large enough—but finite—time point t, for which convergence in probability does hold.

In practice, steady-state rewards are estimated through stochastic simulation using the method of batch means (e.g., see [27], [28]). Equilibrium conditions for the ODE model are based on two criteria that are checked during numerical integration. Recalling that the numerical solution of an ODE is given as a *mesh* of time points $[t_0, t_1, \ldots, t_n]$ and corresponding solution vectors $[x_0, x_1, \ldots, x_n]$ (e.g., [29]), one criterion checks if the norm of the derivative of the current solution vector x_i is less than a given *absolute* threshold. The other criterion checks whether the norm of the difference between two subsequent solution vectors $|x_i - x_{i-1}|$, with $i \geq 1$, is less than a given *relative* threshold. If both criteria are satisfied, then the ODE is said to reach equilibrium at time t_i and the solution vector x_i is used to approximate the fluid steady-state rewards. In the analyses conducted in this paper both thresholds were set to 10^{-6}.

4 Action Throughput

Throughput is a performance metric which has counterparts in other formalisms for quantitative evaluation. In queueing theory, it is associated with a station and denotes the frequency of service; in stochastic Petri nets, it indicates the frequency of firing of a transition. In stochastic process algebras, throughput measures the frequency of execution of an action type.

Action throughput is introduced in [6] for the original Markovian interpretation of the language, which maps a PEPA component P_k onto a state of the underlying CTMC. For a probability distribution $\pi(t)$ of the CTMC, the throughput of an action type $\alpha \in \mathcal{A}$ is defined as

$$\sum_k \pi_k(t)\, t_k, \text{ where } t_k = \sum_{(\alpha, r) \in \mathcal{A}ct(P_k)} r.$$

$\mathcal{A}ct(P_k)$ denotes the set of activities enabled by component P_k. The reward sums over all the rates of the activities which are labelled with the action type α. An equivalent formulation for the population-based CTMC may be given in terms of the generating functions. For some action type α, each of the generating functions $\varphi_\alpha(\xi, l)$ gives the frequency of some activity of that type occurring, which changes the population counts corresponding to the nonzero elements of the vector l. Therefore, summing over all such generating functions gives the throughput of interest for each state ξ.

Definition 1. *The reward function for the action throughput of $\alpha \in \mathcal{A}$, denoted by $Th_\alpha(\omega)$, is*

$$Th_\alpha(\omega) = \sum_{l \in \mathbb{Z}^d} \varphi_\alpha\left(\omega, l\right).$$

The generic argument ω is intended to be $X_n(t)/n$ for the Markovian reward, and $x(t)$ for its deterministic approximation. Therefore, the deterministic approximation of the throughput of action α is

$$Th_\alpha(x(t)) = \sum_{l \in \mathbb{Z}^d} \varphi_\alpha\left(x\left(t\right), l\right).$$

Convergence in probability is satisfied by throughput rewards because of their Lipschitz continuity. In the remainder of this section, we show how throughput enjoys a stronger notion of convergence, i.e., convergence in mean (written $\xrightarrow{\mathbb{E}}$):

$$\lim_{n \to \infty} \mathbb{E}\left[Th_\alpha(X_n(t)/n) - Th_\alpha(x(t))\right] = 0, \text{ for any } t \text{ and } \alpha.$$

A sufficient condition for convergence in mean of a succession of random variables which enjoy convergence in probability is provided by the following

Theorem 2 (Dominated Convergence)**.** *If $Y_n \xrightarrow{\mathbb{P}} Y$ and Y_n is uniformly bounded, i.e., there exists M such that $|Y_n| < M$ almost surely, then $Y_n \xrightarrow{\mathbb{E}} Y$.*

Theorem 3. *Let $\{X_n(t)\}$ be the sequence of random variables of the density process of a PEPA model, for any fixed t. Let $\alpha \in \mathcal{A}$. If Th_α is continuous at $x(t)$ then,*

$$Th_\alpha\left(X_n(t)/n\right) \xrightarrow{\mathbb{E}} Th_\alpha(x(t)).$$

Proof: Convergence in probability of $Th_\alpha\left(X_n(t)/n\right)$ follows from the fact that Th_α is expressed as a summation of generating functions, which are Lipschitz continuous. Therefore, in order to prove convergence in mean it is sufficient to check for uniform boundedness of $Th_\alpha\left(X_n(t)/n\right)$. Using the same arguments as in Theorem 2 of [5], the family of CTMCs $\{X_n(t)\}$ is such that a coordinate $\xi_{i,j}$ of the population vector for the n-th CTMC takes values in $\left\{0, 1, \ldots, \sum_{k=1}^{N_i} n\,\delta_{i,k}\right\}$, hence $\{X_n(t)\}$ may be bounded by a closed (d-dimensional) interval which depends on δ (and does not depend on n). On that interval the Extreme Value Theorem (e.g. [30], Theorem 11.22), holds because of the continuity of Th_α. Therefore, $Th_\alpha(X_n(t)/n)$ is also bounded, as required to complete the proof. □

The property in (8) is trivially satisfied because it holds for the generating functions, as shown in (10). Therefore, it holds that $Th_\alpha(X_n(t)/n) = Th_\alpha(X_n(t))/n$ for any $\alpha \in \mathcal{A}$ and $n \in \mathbb{N}$. With regard to the model in (1), the following reward functions are defined:

$$\begin{aligned} Th_{think}(\omega) &= r_2\omega_2 \\ Th_{log}(\omega) &= r_4\omega_4 \\ Th_{transfer}(\omega) &= \min\left(r_1\omega_1, r_3\omega_3\right). \end{aligned}$$

These equations may be used, for instance, to reveal the difference in the behaviours of the two models illustrated in Fig. 2—in particular, given the steady-state regime, the faster model has twice the throughput of the slower one.

4.1 Location-Aware Throughput

According to Definition 1, throughput is a system-related measure as it does not take account of the identity of the sequential components involved. However, in PEPA distinct components can engage in activities of the same type independently from each other. The formulation of throughput can be refined so as to include location (i.e., identity) awareness and to restrict the estimation of throughput to a subset of components $C_{i,j}$ in the system. Let $\overline{\mathcal{C}}$ be such a subset, $L(\overline{\mathcal{C}})$ gives the subset of jumps l related to transitions in which the elements of $\overline{\mathcal{C}}$ are involved. Such transitions are obtained by considering all the jumps l for which -1 is present in one of the coordinates in the population vector corresponding to the derivatives in $\overline{\mathcal{C}}$. As observed above, $l_{i,j} = -1$ indicates that the population of the component $C_{i,j}$ is decreased by one because of the transition, i.e., the activity is being performed by the component. Thus,

$$L(\overline{\mathcal{C}}) = \left\{l \in \mathbb{Z}^d : l_{i,j} = -1 \wedge C_{i,j} \in \overline{\mathcal{C}}\right\}. \tag{12}$$

The location-aware throughput of α with respect to $\overline{\mathcal{C}}$, denoted by $Th_\alpha(\omega \,|\, \overline{\mathcal{C}})$, is

$$Th_\alpha(\omega \,|\, \overline{\mathcal{C}}) = \sum_{l \in L(\overline{\mathcal{C}})} \varphi_\alpha\left(\omega, l\right).$$

In addition to preserving continuity, it is straightforward to see that, for any $\overline{\mathcal{C}}$ and α, $Th_\alpha(\omega \,|\, \overline{\mathcal{C}}) \leq Th_\alpha(\omega)$, for any ω.

Location-aware throughput is not useful in (1) because any sequential component is always involved in the activities which it enables. For instance, the action *transfer* is carried out by both *Upload* and *Download*, and the sets of independent actions enabled by the two components are disjoint. Suppose now that the definition of *Log* in (1) is replaced with

$$Log \stackrel{def}{=} (think, r_4).Upload,$$

i.e., the action type *log* is replaced by *think*. This gives rise to an identical underlying system of differential equations although the generating function associated with the above definition is now $\varphi_{think}\big(\xi, (0,0,1,-1)\big) = r_4\xi_4$ in place of $\varphi_{log}\big(\xi, (0,0,1,-1)\big) = r_4\xi_4$. Since the action set in the cooperation operator is not changed, the activities *think* performed by *Think* and *Log* are carried out without synchronisation. In this modified model, the location-aware throughputs of action *think* are

$$\begin{aligned} Th_{think}\big(\omega \,|\, \{Download, Think\}\big) &= r_2\omega_2 \\ Th_{think}\big(\omega \,|\, \{Upload, Log\}\big) &= r_4\omega_4 \end{aligned}$$

and

$$\begin{aligned} Th_{think}(\omega) = Th_{think}\big(\omega \,|\, \{Download, Think\}\big) \\ + Th_{think}\big(\omega \,|\, \{Upload, Log\}\big). \end{aligned}$$

Another useful application of location-aware throughput is in cases where there are two components performing a shared action with a third component, *independently* from each other. Consider for instance the following definition of another client

$$\begin{aligned} SuperUser &\stackrel{def}{=} (transfer, r_5).Rest \\ Rest &\stackrel{def}{=} (rest, r_6).SuperUser \end{aligned}$$

and the system equation

$$(Download[N_C] \parallel SuperUser[N_R]) \underset{\{transfer\}}{\bowtie} Upload[N_S].$$

(An analogous scenario will be discussed in more detail in Section 7.) The components *Download* and *SuperUser* will be mapped onto two distinct coordinates in the population vector representation. The empty cooperation set between them indicates no cooperation, but each component will independently perform the action *transfer* cooperatively with *Upload*.

In this case the estimates $Th_{transfer}(\omega \mid \{Download\})$ and $Th_{transfer}(\omega \mid \{SuperUser\})$ disaggregate the overall throughput $Th_{transfer}$ into the throughputs of two constituting interactions between $Download$ and $Upload$, and between $SuperUser$ and $Upload$.

Location-aware throughput becomes counterintuitive if applied to components exhibiting *self-loops*, i.e., when they do not change behaviour after performing an action. For instance, in a model containing $S \stackrel{def}{=} (\alpha, r).S$, it holds that $Th_\alpha(\omega \mid \{S\}) = 0$ because the element of l for S is equal to $-1+1=0$. This situation could be in principle improved by a slight modification of the fluid semantics so as to collect two vectors, l^- and l^+, respectively recording the negative and the positive contributions to the jump vector l, and let $l = l^- + l^+$. With this change, the throughput would be defined through l^-, which would record a nonzero entry even for self-loops.

5 Capacity Utilisation

Capacity utilisation is a performance metric which may be associated with a sequential component to indicate the proportion of time that it engages in some activity, either independently or in synchronisation with other components. This is analogous to the definition of utilisation in queueing networks, which denotes the proportion of time that a service centre serves a customer. This section gives an informal interpretation of capacity utilisation in PEPA, presents its definition with respect to the framework developed in Section 3, and applies this notion to the running example.

5.1 Motivation

The question of how often a device is utilised in a system arises frequently in performance studies. A device that is under-utilised may represent wasteful consumption of resources, whilst devices with utilisation close to unity may indicate overload and a bottleneck which affects the system's overall behaviour. For instance, let us consider the model in (1), and suppose that the modeller is interested in the utilisation of the user component $\mathcal{C}_1 = \{Download, Think\}$ (similarly, the server component is defined as $\mathcal{C}_2 = \{Upload, Log\}$). For simplicity, let us consider the simple case $N_C = N_S = 1$, whose state space representation was shown in Fig. 1.

It is interesting to note that although the sub-vector for $\mathcal{C}_1$ is the same in states 1 and 3, the behaviour of the two states is profoundly different. In 1, both $\mathcal{C}_1$ and $\mathcal{C}_2$ enable $transfer$, whereas in 3 the activity cannot be carried out because it is not enabled by $\mathcal{C}_2$. Similar considerations apply with respect to the behaviour of $\mathcal{C}_2$. In this case, $(1,0)$ is the same sub-vector in states 1 and 4 although action $transfer$ cannot be carried out in 4 because it is not enabled by $\mathcal{C}_1$. Therefore, an intuitive requirement for the notion of capacity utilisation is that it take account of these different dynamic behaviours across the state space. In addition, it is also natural to assign a unitary capacity utilisation to independent actions, to capture the observation that they can always be performed when locally enabled and their execution is not dependent upon the behaviour of other components of the system.

Let $CU_{\mathcal{C}_1}(k)$ denote the capacity utilisation of $\mathcal{C}_1$ in state k of the population-based CTMC. A rather crude reward structure may be the following:

$$CU_{\mathcal{C}_1}(k) = \begin{cases} 1 & \text{if } k = 1, 2, 4, \\ 0 & \text{if } k = 3. \end{cases} \tag{13}$$

where 1 is assigned to state 1 because the shared action $transfer$ can be performed, and to states 2 and 4 because $\mathcal{C}_1$ is engaged in the independent action $think$. However, this definition fails to account for potential under-utilisation arising from the execution of $transfer$. The definition of $Download$ may be interpreted as that of a component which can perform the action at the maximum rate of r_1. According to the semantics of PEPA, the corresponding transition from state 1 to state 2 occurs at the rate $\min(r_1, r_3)$. Therefore, the value $\min(r_1, r_3)/r_1$ seems better suited to measure the fraction of the upload capacity of $\mathcal{C}_1$ that is consumed in state 1.

In general, in order to refine (13), the reward may assign a fraction to each state of the CTMC. The numerator of this fraction measures the total activity rate enabled, whereas the denominator indicates the maximum rate exhibited by the component. This latter quantity corresponds to the component's *parametric apparent rate*, denoted by $r_\alpha(\cdot)$, as introduced in [5]. Thus, the unitary values of capacity utilisation for states 2 and 4 may be interpreted as the fraction r_2/r_2. Clearly, independent actions are always assigned unitary utilisation. Hence, (13) can be revised as

$$CU_{\mathcal{C}_1}(k) = \begin{cases} \min(r_1, r_3)/r_1 & \text{if } k = 1, \\ r_2/r_2 = 1 & \text{if } k = 2, 4, \\ 0 & \text{if } k = 3. \end{cases}$$

Notice that the fraction $\min(r_1, r_3)/r_3$ could be analogously assigned to state 1 for the computation of the capacity utilisation of $\mathcal{C}_2$:

$$CU_{\mathcal{C}_2}(k) = \begin{cases} \min(r_1, r_3)/r_3 & \text{if } k = 1, \\ r_4/r_4 = 1 & \text{if } k = 2, 3 \\ 0 & \text{if } k = 4. \end{cases}$$

5.2 General Definition and Properties

The following reward function extends the definition of capacity utilisation to the population-based representation of an arbitrary PEPA model.

Definition 2 (Capacity Utilisation). *Let $\mathcal{C}_i$ denote a derivative set in the reduced context with N_i distinct derivatives $C_{i,1}, C_{i,2}, \ldots, C_{i,N_i}$. The capacity utilisation of $\mathcal{C}_i$, denoted by $CU_{\mathcal{C}_i}$, measures the proportion of time that the derivatives of $\mathcal{C}_i$ are engaged in some action:*

$$CU_{\mathcal{C}_i}(\omega) = \frac{\sum_{\alpha \in \mathcal{A}} \sum_{l \in L(\mathcal{C}_i)} \varphi_\alpha(\omega, l)}{\sum_{\alpha \in \mathcal{A}} \sum_{j=1}^{N_i} r_\alpha(C_{i,j}) \omega_{i,j}}$$

where L is defined as in (12).

The numerator of Definition 2 gives the *overall utilised capacity* by the components which exhibit the local states in $\mathcal{C}_i$. Similarly, the denominator provides the *overall available capacity* of all such components, as it sums across the apparent rates of all local states, for all action types enabled. Thus, the fraction measures the capacity of $\mathcal{C}_i$ that is utilised by the system. The following proposition restates Theorem 1 for capacity utilisation.

Proposition 1. *If $CU_{\mathcal{C}_i}$ is continuous at $x(t)$ then $CU_{\mathcal{C}_i}\big(X_n(t)/n\big) \xrightarrow{\mathbb{P}} CU_{\mathcal{C}_i}\big(x(t)\big)$.*

To show that capacity utilisation satisfies (8), write $CU_{\mathcal{C}_i}$ explicitly as a fraction between two functions $N(\omega)$ and $D(\omega)$ which satisfy the condition in (8):

$$\begin{aligned} CU_{\mathcal{C}_i}(X_n(t)/n) &= \frac{N(X_n(t)/n)}{D(X_n(t)/n)} \\ &= \frac{N(X_n(t))/n}{D(X_n(t))/n} = CU_{\mathcal{C}_i}(X_n(t)). \end{aligned}$$

Convergence in mean cannot be proven using the arguments of Theorem 3 because $CU_{\mathcal{C}_i}$ is not continuous in the zero vector in $\mathbb{R}^d$. However, the reward function is continuous at all values taken by $x(t)$. To show this, notice that $CU_{\mathcal{C}_i}$ is a rational function of two Lipschitz-continuous functions. Thus, it is sufficient to establish that $\sum_{\alpha\in\mathcal{A}}\sum_{j=1}^{N_i} r_\alpha(C_{i,j})x_{i,j}(t) > 0$ for all t. But at least one coordinate of $x(t)$ must be strictly positive, because of the conservation law discussed in Section 2. Let $x_{i,j}$ be such a coordinate. The corresponding component $C_{i,j}$ must enable at least one action type α, which yields $r_\alpha(C_{i,j}) > 0$.

In the running example, the capacity utilisations of two sequential components are

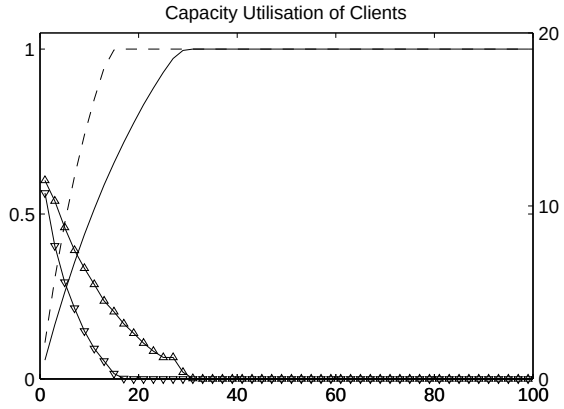
$$CU_{\mathcal{C}_1}(\omega) = \frac{\min(r_1\omega_1, r_3\omega_3) + r_2\omega_2}{r_1\omega_1 + r_2\omega_2}, \tag{14}$$

$$CU_{\mathcal{C}_2}(\omega) = \frac{\min(r_1\omega_1, r_3\omega_3) + r_4\omega_4}{r_3\omega_3 + r_4\omega_4}. \tag{15}$$

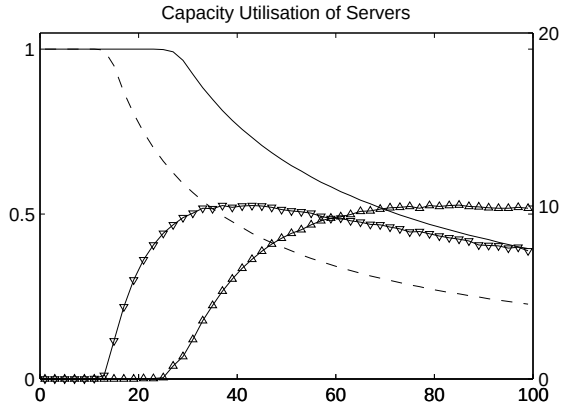
5.3 Numerical Example

As a practical application, Fig. 3a plots the results of a sensitivity analysis of the steady-state capacity utilisation (14) with respect to the parameters r_3 and N_S. Two values for the rate r_3 were considered, i.e., 1.0 and 2.0 (solid and dashed lines, respectively), and N_S was varied between 1 and 100. All the other parameters of the system were set as follows: $r_1 = 1.0, r_2 = 10.0, r_4 = 50.0$ and $N_C = 30$. Here, the capacity utilisation of the clients (in Fig. 3a) increases with N_S because the thirty clients are increasingly likely to find servers to download from. Clearly, for large N_S the probability of finding an available server component is so high that the capacity of the clients is fully utilised.

Figure 3b shows the same sensitivity analysis for (15). Qualitatively, the trajectories of the curves for the two values of r_3 are in agreement with the intuition that, as N_S increases, each of the sequential components is less utilised on average. A particularly interesting point is $N_S = 30$, i.e., there are as many clients as server components. When $r_3 = 1.0$ they have the same rate for the shared action, and the high capacity utilisation (i.e., 0.992) obtained in this case highlights that each pair is very likely to be engaged in a synchronised activity. However, the same model with $r_3 = 2.0$ yields a capacity utilisation of about 50%—this is explained by the fact that the capacity of servers is twice that of clients.



(a)



(b)

Fig. 3: Capacity utilisations for (1) as a function of N_S. Left y-axis: Markovian analysis (solid line for estimates with $r_3 = 1.0$, dashed line for those with $r_3 = 2.0$). Right y-axis: Percentage errors of the differential estimates (upward-pointing triangles: models with $r_3 = 1.0$, downward-pointing triangles: models with $r_3 = 2.0$).

The error plots in Fig. 3 (cf. lines with markers) show that the accuracy is within 10% in most cases, and that higher capacity utilisations are usually approximated better. Despite these being reasonable estimates in practical situations, it is nevertheless worthwhile pointing out that the model instances considered in this simple example do not necessitate deterministic approximation because their state space sizes—ranging from 62 to 3131 states—were well within the reach of numerical CTMC solvers. As will be shown in Section 7, much greater accuracy is typically achieved for models of larger size.

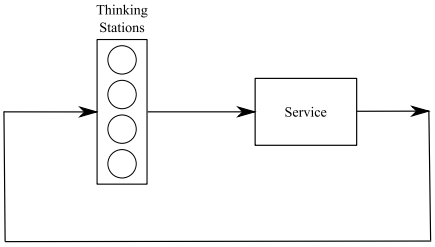


Fig. 4: Schematic representation of the system used for the application of Little's law to PEPA models

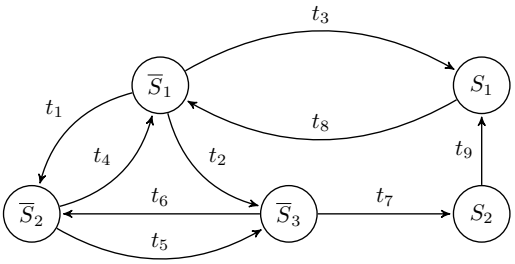


Fig. 5: Derivation graph of a sequential component. The local derivatives are partitioned into $S = \{S_1, S_2\}$ and $\overline{S} = \{\overline{S}_1, \overline{S}_2, \overline{S}_3\}$, interpreted as the component being inside and outside the system, respectively. Thus, transitions t_3 and t_7 are paths of entry into the system. Conversely, the system is exited via t_8.

6 Average Response Time

Throughput and capacity utilisation are meaningful performance metrics at every time point of the system. Indeed, it is possible to define the notion of *peak* and *minimum* across a finite time interval, or to normalise these metrics with respect to the time-frame of interest, as outlined in Section 1. Instead, the notion of average response time discussed in this section can only be applied to systems under equilibrium conditions because it is based on Little's law [31], providing the response time as a function of a specific kind of location-aware throughput and of the steady-state population levels of the model's sequential components.

6.1 Little's Law

In its general formulation, Little's law considers a system under steady-state conditions with L users, arriving at rate λ and subject to an average waiting time W. The law states that

$$L = \lambda W. \tag{16}$$

Here, this relation is used to determine $W = L/\lambda$, i.e., the average response time is estimated from the computation of population levels and action throughputs, which can be obtained as discussed in the previous sections. A slightly simpler formulation of Little's law requires the computation of only one estimate and may be applied for closed systems such as in Fig. 4. The system comprises a total population of N users. The arrival rate for service is λ, the average service time is W, and each user spends some time Z between successive admissions into the system. Under steady-state conditions, the following holds

$$N = \lambda\,(Z + W) \tag{17}$$

which can be used to give $W = N/\lambda - Z$. Note that (17) is obtained by applying (16) to the system comprising the thinking stations and the service, observing that the waiting time is the sum of the average waiting times in the two sub-systems. This expression requires the calculation of λ, since N and Z are model parameters.

The PEPA model of (1) can be thought of as an instance of the system considered in Fig. 4. The thinking stations are represented by the number of components which exhibit the local derivative $Think$, whilst the service centre comprises the components which exhibit the local derivative $Download$. The total number of users is $N = N_C$, and the average thinking time $Z = 1/r_2$. Finally, the arrival rate at the service centre λ is calculated as the steady-state action throughput of $think$. Thus, the average response time can be calculated as follows

$$W = \frac{N_C}{Th_{think}(\infty)} - \frac{1}{r_2}.$$

Such a syntactic structure of the user component has been assumed in previous work on this topic (e.g., [18]), though it cannot be used for more complex user descriptions. For instance, Fig. 5 shows the derivation graph of one such sequential component, in which multiple paths of entry and exit are defined. The following section is concerned with the development of a general formulation for the average response time which does not require any assumption for the applicability of the analysis.

6.2 General Formulation

Let $\mathcal{C}_i$ be the component of the reduced context representing the user with respect to whom the average response time is to be computed. Let $S^i \subset \mathcal{C}_i, S^i \neq \emptyset$ be the subset of derivatives which indicate the presence of the user in the system, S^i induces a binary partition $\{S^i, \overline{S^i}\}$. The derivatives in $\overline{S^i}$ denote the states in which the user is outside the system. Let μ_i^l and $\overline{\mu}_i^l$ be the subsets of the jump vector l corresponding to the population levels of S^i and $\overline{S^i}$, respectively. By the population-based semantics of PEPA, the number of non-zero elements in $\mu_i^l \cup \overline{\mu}_i^l$ can be either zero or two. There cannot be only one non-zero element because this would imply an increase (resp., decrease) in the population level of some derivative without a corresponding decrease (resp., increase) in the population level of some other derivative. However, this is clearly not allowed by the fact that the derivation graphs of the sequential components are strongly connected—the dynamic creation or destruction of sequential components is not possible. These non-zero elements must be -1 and $+1$, because the transition records unitary changes in the population levels. Thus,

TABLE 2: The set of subvectors μ_i^l and $\overline{\mu}_i^l$ for the sequential component in Fig. 5. Transitions t_3 and t_7 indicate the entry of a user into the system, because a population level in μ_i^l is incremented by one and, correspondingly, a population level in $\overline{\mu}_i^l$ is decreased by one.

Transition	μ_i^l		$\overline{\mu}_i^l$		
	S_1^i	S_2^i	$\overline{S^i}_1$	$\overline{S^i}_2$	$\overline{S^i}_3$
t_1	0	0	-1	$+1$	0
t_2	0	0	-1	0	$+1$
$\mathbf{t_3}$	$\mathbf{+1}$	0	$\mathbf{-1}$	0	0
t_4	0	0	$+1$	-1	0
t_5	0	0	0	-1	$+1$
t_6	0	0	0	$+1$	-1
$\mathbf{t_7}$	0	$\mathbf{+1}$	0	0	$\mathbf{-1}$
t_8	-1	0	$+1$	0	0
t_9	$+1$	-1	0	0	0

there are five cases according to the location of the non-zero elements:

- $\{-1,+1\} \notin \mu_i^l \cup \overline{\mu}_i^l$ indicates a jump in which the population levels of $\mathcal{C}_i$ are not affected (for instance, an independent action performed by some other sequential component in the system).
- $\{-1,+1\} \in \mu_i^l$ indicates a transition within the system in which the user is engaged.
- $\{-1,+1\} \in \overline{\mu}_i^l$ is the symmetric case in which user is engaged, though the activity takes place outside the system.
- $\{-1\} \in \mu_i^l$ and $\{+1\} \in \overline{\mu}_i^l$ represents the departure of one user from the system, as the population level of some component in S^i is decreased by one, with a corresponding increase observed for the population of some component in $\overline{S^i}$.
- $\{-1\} \in \overline{\mu}_i^l$ and $\{+1\} \in \mu_i^l$ is the subset of jumps of interest with respect to the computation of the average response time, as it represents the arrival of users into the system. The population level of some component in S^i is increased by one, and, correspondingly, the population of some component in $\overline{S^i}$ is decreased.

The set of jumps for the sequential component in Fig. 5 is shown in Table 2. The following two definitions specify how to calculate the throughput of arrivals and the average number of users in a PEPA model.

Definition 3. *The throughput of the arrivals of S^i into the system, denoted by λ_{S^i}, is the sum of the throughputs, for all action types, across all transitions such that $\{-1\} \in \overline{\mu}_i^l$ and $\{+1\} \in \mu_i^l$:*

$$\lambda_{S^i}(\omega) = \sum_{\alpha \in \mathcal{A}, \{-1\} \in \overline{\mu}_i^l, \{+1\} \in \mu_i^l} \varphi_\alpha(\omega, l)$$

Definition 4. *The population count of the users in the system, denoted by L_{S^i}, is*

$$L_{S^i}(\omega) = \sum_{C_{i,j} \in S^i} \omega_{i,j}$$

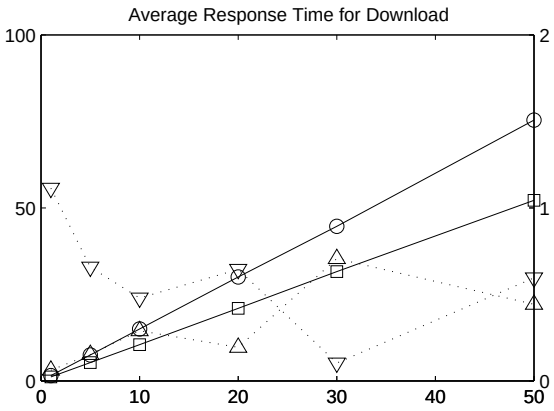


Fig. 6: Average response time calculation for (1) as a function of N_C. Left y-axis: Markovian analysis (circles for the model with $r_4 = 0.2$, squares for the model with $r_4 = 2.0$). Right y-axis: Percentage errors of the differential estimates (upward-pointing triangles: model with $r_4 = 0.2$, downward-pointing triangles: model with $r_4 = 2.0$).

Using the same arguments as in Theorem 3, the following proposition holds.

Proposition 2. *For any $S^i \subset \mathcal{C}_i, S^i \neq \emptyset$, it holds that $\lambda_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} \lambda_{S^i}(x(t))$ and that $L_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} L_{S^i}(x(t))$.*

Based on this result, the following approximation for the calculation of the fluid response time will be used:

$$\frac{L_{S^i}(x(t))}{\lambda_{S^i}(x(t))} \approx \frac{\mathbb{E}\left[L_{S^i}(X_n(t)/n)\right]}{\mathbb{E}\left[\lambda_{S^i}(X_n(t)/n)\right]} = \frac{\mathbb{E}\left[L_{S^i}(X_n(t))\right]/n}{\mathbb{E}\left[\lambda_{S^i}(X_n(t))\right]/n} = \frac{\mathbb{E}\left[L_{S^i}(X_n(t))\right]}{\mathbb{E}\left[\lambda_{S^i}(X_n(t))\right]} \quad (18)$$

where the first equality follows directly from Definitions 3 and 4, and the rightmost fraction corresponds to the definition of average response time for the n-th Markov chain of the family of PEPA models. As stated above, although this calculation is in principle applicable to any time point, it is only meaningful under steady-state conditions.

For the model (1), the partition $S^i = \{Download\}, \overline{S^i} = \{Think\}$ gives rise to the following definitions of L_{S^i} and λ_{S^i} for the average response time of $\mathcal{C}_1$:

$$L_{S^i}(\omega) = \omega_1 \quad (19)$$

$$\lambda_{S^i}(\omega) = r_2\, \omega_2 \quad (20)$$

Figure 6 shows an example of average response time calculation for this model, experimenting with population levels of the clients ranging from 1 to 50 and two values for rate r_4. In all cases N_S was kept fixed at 10, and the rate set as in Section 5 (with $r_3 = 1.0$). As expected, the response time does not change significantly when the population of clients is less than that of servers. In contrast, a dramatic increase is observed when the

number of clients is significantly more than the number of server components. Clearly, increasing r_4 reduces the response time although it does not impact on its qualitative behaviour. In all cases considered in this study, the accuracy of the fluid approximation was mostly within one percent.

7 Numerical Validation

This section presents numerical validations of the performance metrics introduced in the previous sections. It examines a more complex model than (1), which has the following features: (i) use of the choice operator to describe alternative behaviour; (ii) more structured system equation, comprising five sequential components and a pattern of composition similar to that described in Section 4.1; (iii) faster rate of state-space growth; (iv) unlike (1), the average response times cannot be calculated using the simplified version (17).

The model descriptions were parametrised by the activity rates and the population level density. A set of 300 model instances was obtained by assigning randomly chosen rate parameters and initial densities drawn from uniform distributions. The objective of this approach is to measure the quality of the deterministic approximation on a broad spectrum of behaviours, from models which exhibit poor indices (i.e., low capacity utilisation or high average response times) to those with good performance. Each model instance was analysed for three scale factors, $n = 1$, $n = 10$, and $n = 100$.

The validation concerned performance estimates at equilibrium—this is necessary for the computation of average response times whereas steady-state measures were taken as representative conditions for the calculation of throughput and capacity utilisation. All analyses were conducted using the *PEPA Eclipse Plug-in* [32].

7.1 Model Description

This model comprises two distinct classes of users, C_1 and C_2, defined as follows (alongside the definitions are the corresponding coordinates in the population vector):

$$\begin{array}{lrcl}
\xi_1 & C_1_Think & \stackrel{def}{=} & (think, (1-p)\, p_{db}\, t_1).C_1_UseDb \\
 & & + & (think, (1-p)\, p'_{db}\, t_1).C_1_UseCpu \\
 & & + & (think, p\, t_1).C_1_Think' \\
\xi_2 & C_1_UseCpu & \stackrel{def}{=} & (useCpu, c_1).C_1_Think \\
\xi_3 & C_1_UseDb & \stackrel{def}{=} & (useDb, d_1\, c_1).C_1_Think \\
\xi_4 & C_1_Think' & \stackrel{def}{=} & (think, t'_1).C_1_UseCpu
\end{array}$$

$$\begin{array}{lrcl}
\xi_5 & C_2_Think & \stackrel{def}{=} & (think, q_{db}\, t_2).C_2_UseDb \\
 & & + & (think, q'_{db}\, t_2).C_2_UseCpu \\
\xi_6 & C_2_UseCpu & \stackrel{def}{=} & (useCpu, c_2).C_2_Think \\
\xi_7 & C_2_UseDb & \stackrel{def}{=} & (useDb, d_2\, c_2).C_2_Think
\end{array}$$

The use of the choice operator in C_1_Think and C_2_Think allows the specification of conditional behaviour. The action *think* is performed at rates t_1 and t_2 by C_1 and C_2 respectively. With probability p, C_1 moves into a second thinking state, C_1_Think'. With probability $1-p$ the component may behave either as C_1_UseDb, with probability p_{db}, or as C_1_UseCpu, with probability $p'_{db} = 1 - p_{db}$. The behaviour of C_2 is similar, although the second thinking process is not exhibited. Both components may perform the actions $useDb$ and $useCpu$ although with different local rates. Specifically the rates of $useDb$ are expressed as ratios, i.e., d_1 and d_2, of the rates of $useCpu$. If $d_1 < 1$ then the behaviour of C_1 is such that it requires longer data-bound activities. Conversely, if $d_2 > 1$ then C_2 carries out longer (i.e., slower) CPU-bound operations. The states $UseCpu$ and $UseDb$ of the two classes of components are synchronisation points with the following two-state server components

$$\begin{array}{lrcl}
\xi_8 & Cpu_Execute & \stackrel{def}{=} & (useCpu, c).Cpu_Log \\
\xi_9 & Cpu_Log & \stackrel{def}{=} & (log, l_c).Cpu_Execute
\end{array}$$

$$\begin{array}{lrcl}
\xi_{10} & Db_Execute & \stackrel{def}{=} & (useDb, d).Db_Log \\
\xi_{11} & Db_Log & \stackrel{def}{=} & (log, l_d).Db_Execute
\end{array}$$

The action type *log* represents a synchronising activity with the component

$$\xi_{12} \quad Logger\_Log \stackrel{def}{=} (log, l).Logger\_Log$$

The resource-sharing nature of this activity is captured by the composition

$$(Cpu\_Execute \parallel Db\_Execute) \underset{\{log\}}{\bowtie} Logger\_Log \tag{21}$$

Finally, the description of the whole system under study combines (21) with the user components

$$\begin{aligned}
System \stackrel{def}{=} & (C_1_Think[N_{C_1}] \parallel C_2_Think[N_{C_2}]) \\
& \underset{\{useCpu, useDb\}}{\bowtie} \big((Cpu_Execute[N_C] \parallel Db_Execute[N_D]) \\
& \qquad \underset{\{log\}}{\bowtie} Logger_Log[N_L]\big)
\end{aligned} \tag{22}$$

The densities used for this validation were of the form $(A, 0, 0, 0, B, 0, 0, C, 0, D, 0, E)$, where A, B, C, D, E were chosen randomly in $\{1, \ldots 10\}$. The rate parameters were drawn from uniform distributions in $[0.1, 50]$. The ratios d_1 and d_2 were drawn from uniform distributions in $[0, 1]$ and $[1, 10]$, respectively. The following reward functions were used for the validation:

$$\begin{aligned}
Th_{useCpu}(\omega) &= \min(c_1\omega_2 + c_2\omega_6, c\omega_8) \\
CU_{Cpu}(\omega) &= \frac{\min(c_1\omega_2 + c_2\omega_6, c\omega_8) + l_c\omega_9}{c\omega_8 + l_c\omega_9} \\
Th_{useDb}(\omega) &= \min(d_1\, c_1\omega_3 + d_2\, c_2\omega_7, d\omega_{10}) \\
CU_{Db}(\omega) &= \frac{\min(d_1\, c_1\omega_3 + d_2\, c_2\omega_7, d\omega_{10}) + l_d\omega_{11}}{d\omega_{10} + l_d\omega_{11}} \\
L_{C_1}(\omega) &= \omega_2 + \omega_3 \\
\lambda_{C_1}(\omega) &= (1-p)t_1\omega_1 + t'_1\omega_4 \\
L_{C_2}(\omega) &= \omega_6 + \omega_7 \\
\lambda_{C_2}(\omega) &= t_2\omega_5
\end{aligned} \tag{23}$$

where $C_i = \{C_i\_UseCpu, C_i\_UseDb\}, i = 1, 2$. L_{C_1} and λ_{C_1} (respectively, L_{C_2} and λ_{C_2}) are combined as in (18) for the calculation of the average response time W_{C_1} (respectively, W_{C_2}). The throughput measures refer to the aggregate throughput of the actions $useCpu$ and $useDb$, but similar results could be obtained by considering the location-aware throughputs of the two distinct classes of users.

The model is computationally demanding for large scale factors. For instance, the state space size when the population levels are all set to one is 48 states, whereas it is over one million states for $N_{C_1} = 10, N_{C_2} = N_D = N_C = N_L = 8$. This suggests that the solution of the CTMC based on explicit state-space enumeration is infeasible for scale factors $n = 10$ and $n = 100$. For this reason, stochastic simulation was used instead. Each model instance was simulated until the 95% confidence intervals of the equilibrium distribution dropped below 1% of the statistical mean.

7.2 Numerical Results

Table 3 shows the statistics (i.e., 5% quantile, average, median, and 95% quantile) for the percentage relative approximation errors for each of the performance indices in (23). Equation (11) was used to compute the errors. Overall, the results show that each statistic decreases as a function of n. For $n = 1$, there are model instances which lead to particularly high errors, however the accuracy of the approximation is already good for $n = 10$, and mostly within 1% for $n = 100$. For all n and for each performance index, the median error is smaller than the average value, indicating that the distribution is shifted towards smaller values. In particular, in some cases models were found to have extremely good accuracy, as indicated by the generally low values for the 5% quantiles, even for $n = 1$. For comparison, the approximation errors for some of the model components are shown in Table 4. These results indicate that the accuracy of the performance indices is qualitatively similar, since the error statistics are of comparable magnitude in all cases.

A further inspection of the validation data set reveals other reasonably predictable properties of the fluid approximation. One property is that, for a given model instance, knowing the quality of the approximation for a specific performance index is not indicative of the behaviour of the model with respect to other indices. For instance, the model instance which gave the worst accuracy for W_{C_2} at $n = 1$ (i.e., about 140%) had an error of about 9% for W_{C_1} and an error of about 5% for CU_{Cpu}. On the other hand, there were model instances which gave bad accuracy in two or more performance indices. For example, the two worst models for Th_{useCpu} (with errors of about 100% and 53%, respectively) were also the worst cases for Th_{useDb} (with errors of about 100% and 44%).

Another noteworthy property of this approximation regards the rate of convergence. There were models which showed a particularly fast convergence rate for some of the performance indices. For instance, the model with the worst accuracy for Th_{useCpu} at $n = 1$ had an approximation error of about 0.023% for $n = 100$, which is a value close to the median error (0.016%, see Table 3). Other model instances were found to be more resistant to convergence. For example, one model instance had an error of 1.399% for Th_{useCpu} at $n = 1$ (sensibly less than the median value). This instance became then the worst model for Th_{useCpu} at $n = 10$ and $n = 100$ with errors 8.807% and 2.345%, respectively. Incidentally, this also highlights that the error may increase with n. The theoretical results of convergence are only asymptotic and do not exclude nonmonotonic error trends for finite n. However, cases such as this one occurred quite rarely in the validation data set considered in this study. In only five cases, the difference between the approximation error at $n = 10$ and the approximation error at $n = 1$ was substantial, i.e., more than 3%. Cases with marginal (i.e., less than 0.2%) positive differences were more frequent, although these are likely to be due to the fact that the confidence level used for terminating the stochastic simulation algorithm (i.e., 1%) was large compared to those error differences.

8 Conclusion

This paper developed a framework in which a Markovian reward structure for a PEPA model may be related to some real function of the chain's deterministic limit. Results of convergence demonstrate that the approximation is sound asymptotically. This framework has been applied to the definition of three important performance indices: throughput, utilisation, and average response time. Interestingly, these indices are defined as functions of process algebra terms, and their interpretations as Markovian reward and real functions of the fluid limit are directly inferred from the operational semantics of the language. Owing to the formality of the language, these results of convergence are not tied to a particular PEPA model, rather they hold in general. The conditions under which convergence holds appear to be mild, insomuch that further useful metrics may be built upon this framework.

The numerical results presented in this paper confirm the validity of ODE approximation for the estimation of indices of performance in general. The analysis becomes more accurate as the size of the system under scrutiny grows. As with the behaviour of the accuracy for the population levels, this approach is particularly desirable for large-scale systems, where approximation errors are consistently within a few percent.

8.1 Cost of the Analysis

Importantly, this kind of analysis is usually possible at a small fraction of the computational effort required for stochastic evaluation. When large-scale models are to be analysed, the numerical solution of CTMC quickly

TABLE 3: Percentage relative approximation errors for the performance indices over 300 random model instances.

Performance Index	$n = 1$				$n = 10$				$n = 100$			
	5%	*Avg.*	*Median*	95%	5%	*Avg.*	*Median*	95%	5%	*Avg.*	*Median*	95%
Th_{useCpu}	0.105	5.547	2.364	18.857	0.005	0.511	0.137	2.233	$\ll 0.001$	0.066	0.016	0.193
Th_{useDb}	0.053	6.510	3.221	21.343	0.004	0.567	0.142	2.722	0.001	0.068	0.024	0.248
CU_{Cpu}	$\ll 0.001$	8.261	4.559	28.450	$\ll 0.001$	1.264	0.318	6.071	$\ll 0.001$	0.143	0.022	0.531
CU_{Db}	0.017	10.548	7.134	32.889	$\ll 0.001$	1.386	0.577	5.003	$\ll 0.001$	0.154	0.062	0.504
W_{C_1}	0.139	14.776	5.251	47.473	0.006	1.442	0.436	6.940	0.002	0.184	0.050	0.529
W_{C_2}	0.056	10.868	5.792	39.961	0.008	1.442	0.217	6.872	0.002	0.195	0.031	0.618

TABLE 4: Percentage relative approximation errors for the population levels over the model instances of Table 3.

Model Component	$n = 1$				$n = 10$				$n = 100$			
	5%	*Avg.*	*Median*	95%	5%	*Avg.*	*Median*	95%	5%	*Avg.*	*Median*	95%
C_1_Think	0.075	7.503	3.418	29.499	0.005	0.891	0.355	3.661	0.001	0.113	0.038	0.384
C_1_UseCpu	0.021	13.015	3.106	75.269	0.002	2.074	0.177	10.291	$\ll 0.001$	0.410	0.020	0.629
C_1_UseDb	0.111	19.481	6.534	83.890	0.006	4.022	0.366	23.958	0.002	0.553	0.067	1.756
C_2_Think	0.042	7.171	4.235	24.674	0.003	0.656	0.179	2.827	0.001	0.080	0.022	0.322
C_2_UseCpu	0.018	12.483	2.704	59.959	0.002	1.960	0.116	8.150	0.001	0.341	0.015	0.542
C_2_UseDb	0.029	29.713	12.509	92.269	0.003	6.011	0.298	46.506	0.001	0.738	0.035	2.752
$Cpu\_Execute$	0.042	7.112	1.352	37.282	0.001	1.447	0.097	7.821	$\ll 0.001$	0.193	0.008	0.434
$Db\_Execute$	0.005	4.846	0.936	22.383	0.001	0.664	0.041	4.038	$\ll 0.001$	0.089	0.005	0.405

becomes infeasible due to state-space explosion, as illustrated in Section 7.1. Therefore, stochastic simulation represents the most viable route. It is interesting to note that the computational cost of simulation is dependent upon the number of performance indices to be evaluated, hereinafter denoted by K. If transient indices are required, then the method of independent replications is a reasonable simulation algorithm to be used (e.g., [33]). Here, K statistic estimators must be set up and the model must be simulated until it reaches the time point of interest. For each replication, K evaluations (one for each performance index) must be performed. The results must be stored by the estimators for the final computation of distributional statistics such as the expected value. If R replications are required to reach convergence, the sole process of computing the performance indices will require at least $K \times R$ evaluations of the reward structures. If the analysis concerns steady-state indices, then a simulation algorithm based on the method of batch means (or variants thereof) seems more appropriate, as discussed in Section 3.3. In this case, the K evaluations must be performed at every advance of simulated time. If T total time steps are required to reach convergence, then a total of $K \times T$ evaluations must be performed.

In contrast, the fluid analysis is carried out by numerical integration of the initial value problem until a specified time point for transient measures, or over a sufficiently long time period for steady-state measures, as discussed in Section 3.3. In either case, this is done to obtain a solution vector and no computation concerns the evaluation of performance metrics. Only after the solution vector of the ODE is obtained, the computation of K performance metrics requires only K evaluations of the reward structures. In addition, if further performance metrics are to be analysed, the same solution vector can be used for those. In practice, stochastic simulation turned out to have an average runtime approximately four orders of magnitude slower than ODE analysis for the case study of Section 7.

ACKNOWLEDGEMENT

Part of this research has been carried out while M.T. and J.D. were at the University of Edinburgh, with the School of Informatics and the School of Engineering, respectively. J.D. acknowledges the financial support from Mobile VCE.

REFERENCES

[1] A. Clark, S. Gilmore, J. Hillston, and M. Tribastone, *Formal Methods for Performance Evaluation: the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007*. Bertinoro, Italy: Springer-Verlag, May–June 2007, vol. 4486, ch. Stochastic Process Algebras, pp. 132–179.

[2] M. Tribastone and S. Gilmore, "Automatic Translation of UML Sequence Diagrams into PEPA Models," in *Fifth International Conference on the Quantitative Evaluaiton of Systems (QEST 2008)*. Saint-Malo, France: IEEE Computer Society, 14–17 September 2008, pp. 205–214.

[3] ——, "Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile," in *Proceedings of the Seventh International Workshop on Software and Performance (WOSP)*. Princeton, New Jersey, USA: ACM, June 2008.

[4] J. Hillston, "Fluid flow approximation of PEPA models," in *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*. Torino, Italy: IEEE Computer Society Press, Sep. 2005, pp. 33–43.

[5] M. Tribastone, S. Gilmore, and J. Hillston, "Scalable Differential Analysis of Process Algebra Models," 2010, *Transactions on Software Engineering*, pre-print. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2010.82

[6] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[7] J. Meyer, "On evaluating the performability of degradable computing systems," *Computers, IEEE Transactions on*, vol. C-29, no. 8, pp. 720–731, Aug. 1980.

[8] R. Smith, K. Trivedi, and A. Ramesh, "Performability analysis: measures, an algorithm, and a case study," *Computers, IEEE Transactions on*, vol. 37, no. 4, pp. 406–417, Apr 1988.

[9] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort, "Composite performance and dependability analysis," *Performance Evaluation*, vol. 14, no. 3-4, pp. 197 – 215, 1992.

[10] J. F. Meyer, "Performability: a retrospective and some pointers to the future," *Performance Evaluation*, vol. 14, no. 3-4, pp. 139 – 156, 1992.

[11] M. Beaudry, "Performance-related reliability measures for computing systems," *Computers, IEEE Transactions on*, vol. C-27, no. 6, pp. 540–547, June 1978.

[12] L. T. Wu, "Operational models for the evaluation of degradable computing systems," *SIGMETRICS Performance Evaluation Review*, vol. 11, no. 4, pp. 179–185, 1982.

[13] W. H. Sanders and J. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," *Dependable Computing for Critical Applications*, pp. 215–247, 1990.

[14] B. Haverkort and K. Trivedi, "Specification techniques for Markov reward models," *Discrete Event Dynamic Systems*, vol. 3, no. 2–3, pp. 219–247, July 1993.

[15] A. Aldini and M. Bernardo, "Mixing logics and rewards for the component-oriented specification of performance measures," *Theor. Comput. Sci.*, vol. 382, no. 1, pp. 3–23, 2007.

[16] G. Clark, S. Gilmore, and J. Hillston, "Specifying performance measures for PEPA," in *ARTS*, ser. Lecture Notes in Computer Science, J.-P. Katoen, Ed., vol. 1601. Springer, 1999, pp. 211–227.

[17] J. T. Bradley, R. Hayden, W. J. Knottenbelt, and T. Suto, "Extracting response times from fluid analysis of performance models," in *SIPEW '08*. Springer-Verlag, 2008, pp. 29–43.

[18] A. Clark, A. Duguid, S. Gilmore, and M. Tribastone, "Partial Evaluation of PEPA Models for Fluid-Flow Analysis," in *EPEW*, ser. LNCS, vol. 5261, 2008, pp. 2–16.

[19] A. Clark, A. Duguid, S. Gilmore, and J. Hillston, "Espresso, a Little Coffee," in *Process Algebra and Stochastically Timed Activities (PASTA)*, Edinburgh, UK, 2008.

[20] M. Tribastone, "Scalable Analysis of Stochastic Process Algebra Models," Ph.D. dissertation, School of Informatics, The University of Edinburgh, 2010.

[21] P. Pollet, "On a model for interference between searching insect parasites," *J. Austral. Math. Soc. Ser. B*, vol. 32, no. 2, pp. 133–150, 1990.

[22] P. Billingsley, *Probability and Measure*, 3rd ed. Wiley, 1995.

[23] J. Ding, "Structural and fluid analysis for large scale PEPA models—with applications to content adaptation systems," Ph.D. dissertation, School of Engineering, The University of Edinburgh, 2010.

[24] R. Darling and J. Norris, "Differential equation approximations for Markov chains," *Probability Surveys*, vol. 5, pp. 37–79, 2008.

[25] J. Kemeny and J. Snell, *Finite Markov Chains*. Van Nostrand, 1960.

[26] L. Gurvits and J. Ledoux, "Markov property for a function of a Markov chain: A linear algebra approach," *Linear Algebra and its Applications*, vol. 404, pp. 85–117, 2005.

[27] B. R. Haverkort, *Performance of computer communication systems: a model-based approach*. John Wiley & Sons, 1998.

[28] C. Alexopoulos and D. Goldsman, "To batch or not to batch?" *ACM Transactions on Modeling and Computer Simulation*, vol. 14, no. 1, pp. 76–114, 2004.

[29] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1988.

[30] P. Fitzpatrick, *Advanced Calculus*, 2nd ed. AMS Bookstore, 2009.

[31] J. Little, "A Proof of the Queuing Formula: $L = \lambda W$," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.

[32] M. Tribastone, A. Duguid, and S. Gilmore, "The PEPA Eclipse Plug-in," *Performance Evaluation Review*, vol. 36, no. 4, pp. 28–33, March 2009.

[33] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, 2009.

Mirco Tribastone is Assistant Professor at the Institut für Informatik of Ludwig-Maximilians-Universität, Munich. He recently received the PhD from the School of Informatics at the University of Edinburgh. His principal interests are in the quantitative evaluation of computer systems using analytical models. He is the lead developer of the *PEPA Eclipse Plug-in Project* which supports a range of qualitative and quantitative analysis methods for PEPA.

Jie Ding is . . .

Stephen Gilmore joined the University of Edinburgh as a Lecturer in October 1990. He was then promoted to Senior Lecturer and is now a Reader in the School of Informatics. He was the Edinburgh site leader for the Sensoria project which developed performance models of large-scale systems and service-oriented computing. He is a co-investigator with Hillston on the Signal project using stochastic process algebra for biochemical signalling pathway analysis.

Jane Hillston received the PhD degree in computer science in 1994 from the University of Edinburgh where she is currently Professor of Quantitative Modelling. Her principal research interests are in the use of stochastic process algebras to model and analyse dynamic systems. She was awarded the first Roger Needham Award in 2004 by the British Computer Society in recognition of her work on PEPA. She was elected to fellowship of the Royal Society of Edinburgh in 2007.