

SPAs for performance modelling: Lecture 7 — Scalable Analysis

Jane Hillston

LFCS, School of Informatics
The University of Edinburgh
Scotland

16th April 2013



THE UNIVERSITY
of EDINBURGH

Outline

- 1 Fluid Rewards
- 2 Introduction to Simulation
- 3 Simulation in PEPA

Outline

- 1 Fluid Rewards
- 2 Introduction to Simulation
- 3 Simulation in PEPA

Introduction

As defined so far the fluid approximation generates a set of ODEs which record the evolution over time of the population counts of the local states or derivatives within a model.

Introduction

As defined so far the fluid approximation generates a set of ODEs which record the evolution over time of the population counts of the local states or derivatives within a model.

This can be informative and particularly when we look at the complete set of counts, can tell us a lot about the behaviour of the system.

Introduction

As defined so far the fluid approximation generates a set of ODEs which record the evolution over time of the population counts of the local states or derivatives within a model.

This can be informative and particularly when we look at the complete set of counts, can tell us a lot about the behaviour of the system.

Nevertheless we will often want to derive performance measures other than the straightforward utilisations that can be inferred directly from counts.

Fluid Rewards

Just as we use **rewards** to help us to derive performance measures from numerical CTMC analysis, so we also do for performance measures from fluid models.

Fluid Rewards

Just as we use **rewards** to help us to derive performance measures from numerical CTMC analysis, so we also do for performance measures from fluid models.

In particular we define notions of **action throughput**, **capacity utilisation**, and **average response time** as reward structures which may be transparently inferred from the process algebraic description.

Fluid Rewards

Just as we use **rewards** to help us to derive performance measures from numerical CTMC analysis, so we also do for performance measures from fluid models.

In particular we define notions of **action throughput**, **capacity utilisation**, and **average response time** as reward structures which may be transparently inferred from the process algebraic description.

This is underpinned by characterisation of the conditions under which

$$\rho(X_n(t)) \approx \rho(x(t))\rho'(n)$$

when we know that the fluid approximation $X_n(t) \approx nx(t)$ holds and where ρ' is a reward-dependent deterministic function.

The Continuous Mapping Theorem

Our extension of reward structures to fluid models depends on the **Continuous Mapping Theorem**.

Continuous Mapping Theorem

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 $g : \mathbb{R}^d \rightarrow \mathbb{R}^k$ be continuous at c . Then,

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

See for example *P. Billingsley, Probability and Measure, 3rd ed. Wiley, 1995.*

Applying this to rewards

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 .

Applying this to rewards

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 is expressed as a reward $\rho(X_n(t))$. Thus we restrict to reward structures which are not explicitly dependent upon the scaling factor n .

Applying this to rewards

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 is expressed as a reward $\rho(X_n(t))$. Thus we restrict to reward structures which are not explicitly dependent upon the scaling factor n .

i.e., the reward structure ρ must satisfy the condition that there exists some ρ' such that

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

Applying this to rewards

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 is expressed as a reward $\rho(X_n(t))$. Thus we restrict to reward structures which are not explicitly dependent upon the scaling factor n .

i.e., the reward structure ρ must satisfy the condition that there exists some ρ' such that

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

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

$$\rho(X_n(t)) \approx \rho'(n)\rho(x(t)).$$

Example model

Download $\stackrel{\text{def}}{=} (transfer, r_1).Think$

Think $\stackrel{\text{def}}{=} (think, r_2).Download$

Upload $\stackrel{\text{def}}{=} (transfer, r_3).Log$

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

System $\stackrel{\text{def}}{=} Download[N_C] \bowtie_{\{transfer\}} Upload[N_S]$

Example model

$$\text{Download} \stackrel{\text{def}}{=} (\text{transfer}, r_1). \text{Think}$$

$$\text{Think} \stackrel{\text{def}}{=} (\text{think}, r_2). \text{Download}$$

$$\text{Upload} \stackrel{\text{def}}{=} (\text{transfer}, r_3). \text{Log}$$

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

$$\text{System} \stackrel{\text{def}}{=} \text{Download}[N_C] \underset{\{\text{transfer}\}}{\boxtimes} \text{Upload}[N_S]$$

The reduced context of *System* is

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

Example model

$$\text{Download} \stackrel{\text{def}}{=} (\text{transfer}, r_1). \text{Think}$$

$$\text{Think} \stackrel{\text{def}}{=} (\text{think}, r_2). \text{Download}$$

$$\text{Upload} \stackrel{\text{def}}{=} (\text{transfer}, r_3). \text{Log}$$

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

$$\text{System} \stackrel{\text{def}}{=} \text{Download}[N_C] \underset{\{\text{transfer}\}}{\boxtimes} \text{Upload}[N_S]$$

The reduced context of *System* is

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

and the generating functions are defined as follows:

$$\psi_{\text{transfer}}(\xi, (-1, 1, -1, 1)) = \min(r_1 \xi_1, r_3 \xi_3)$$

$$\psi_{\text{think}}(\xi, (1, -1, 0, 0)) = r_2 \xi_2$$

$$\psi_{\text{log}}(\xi, (0, 0, 1, -1)) = r_4 \xi_4$$

Action Throughput

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

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

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

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

It holds that $Th_\alpha(X_n(t)/n) = Th_\alpha(X_n(t))/n$ because of the density dependence of the generating functions.

Action Throughput for the example

$$\text{Download} \stackrel{\text{def}}{=} (\text{transfer}, r_1). \text{Think}$$

$$\text{Think} \stackrel{\text{def}}{=} (\text{think}, r_2). \text{Download}$$

$$\text{Upload} \stackrel{\text{def}}{=} (\text{transfer}, r_3). \text{Log}$$

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

$$\text{System} \stackrel{\text{def}}{=} \text{Download}[N_C] \underset{\{\text{transfer}\}}{\bowtie} \text{Upload}[N_S]$$

$$Th_{\text{think}}(\omega) = r_2 \omega_2$$

$$Th_{\text{log}}(\omega) = r_4 \omega_4$$

$$Th_{\text{transfer}}(\omega) = \min(r_1 \omega_1, r_3 \omega_3).$$

Capacity Utilisation

The notion of **capacity utilisation** captures not just whether a component is *in use* but the proportion of its capacity that is being utilised.

Capacity Utilisation

The notion of **capacity utilisation** captures not just whether a component is *in use* but the proportion of its capacity that is being utilised.

Recall that the apparent rate of a component with respect to an activity type records its capacity to perform activity of that type.

Capacity Utilisation

The notion of **capacity utilisation** captures not just whether a component is *in use* but the proportion of its capacity that is being utilised.

Recall that the apparent rate of a component with respect to an activity type records its capacity to perform activity of that type.

When activities are carried out in cooperation, a component may be slowed down by the bounded capacity of the cooperating component, meaning that its capacity is not fully utilised.

Capacity Utilisation

The notion of **capacity utilisation** captures not just whether a component is *in use* but the proportion of its capacity that is being utilised.

Recall that the apparent rate of a component with respect to an activity type records its capacity to perform activity of that type.

When activities are carried out in cooperation, a component may be slowed down by the bounded capacity of the cooperating component, meaning that its capacity is not fully utilised.

In contrast individual activities always represent 100% utilisation when they are enabled.

Capacity Utilisation

Let \mathcal{C}_i denote a derivative set in the reduced context with N_i distinct derivatives $\mathcal{C}_{i,1}, \mathcal{C}_{i,2}, \dots, \mathcal{C}_{i,N_i}$.

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:

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

where $L(\mathcal{C}_i)$ is subset of jumps that \mathcal{C}_i is involved in.

Capacity Utilisation for the example

$$\text{Download} \stackrel{\text{def}}{=} (\text{transfer}, r_1). \text{Think}$$

$$\text{Think} \stackrel{\text{def}}{=} (\text{think}, r_2). \text{Download}$$

$$\text{Upload} \stackrel{\text{def}}{=} (\text{transfer}, r_3). \text{Log}$$

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

$$\text{System} \stackrel{\text{def}}{=} \text{Download}[N_C] \underset{\{\text{transfer}\}}{\boxtimes} \text{Upload}[N_S]$$

$$CU_{C_1}(\omega) = \frac{\min(r_1\omega_1, r_3\omega_3) + r_2\omega_2}{r_1\omega_1 + r_2\omega_2}$$

$$CU_{C_2}(\omega) = \frac{\min(r_1\omega_1, r_3\omega_3) + r_4\omega_4}{r_3\omega_3 + r_4\omega_4}$$

Average Response Time

As previously the approach to **average response time** is via **Little's Law**.

Average Response Time

As previously the approach to **average response time** is via **Little's Law**.

For an arbitrary PEPA component we assume that the set of derivative \mathcal{C}_i can be partitioned into $\{S_i, \overline{S}_i\}$ denoting local states that are **inside** and **outside** the "system" respectively.

Average Response Time

As previously the approach to **average response time** is via **Little's Law**.

For an arbitrary PEPA component we assume that the set of derivative \mathcal{C}_i can be partitioned into $\{S_i, \overline{S}_i\}$ denoting local states that are **inside** and **outside** the "system" respectively.

Let μ_i^l and $\overline{\mu}_i^l$ be the subsets of the jump vector l corresponding to the population vectors of S_i and \overline{S}_i respectively.

Average Response Time

As previously the approach to **average response time** is via **Little's Law**.

For an arbitrary PEPA component we assume that the set of derivative \mathcal{C}_i can be partitioned into $\{S_i, \bar{S}_i\}$ denoting local states that are **inside** and **outside** the "system" respectively.

Let μ_i^l and $\bar{\mu}_i^l$ be the subsets of the jump vector l corresponding to the population vectors of S_i and \bar{S}_i respectively.

By the conservativeness of PEPA models, these jump vectors $(\mu_i^l \cup \bar{\mu}_i^l)$ must have either zero or two non-zero entries.

Average Response Time

As previously the approach to **average response time** is via **Little's Law**.

For an arbitrary PEPA component we assume that the set of derivative \mathcal{C}_i can be partitioned into $\{S_i, \bar{S}_i\}$ denoting local states that are **inside** and **outside** the "system" respectively.

Let μ_i^l and $\bar{\mu}_i^l$ be the subsets of the jump vector l corresponding to the population vectors of S_i and \bar{S}_i respectively.

By the conservativeness of PEPA models, these jump vectors $(\mu_i^l \cup \bar{\mu}_i^l)$ must have either zero or two non-zero entries.

Two non-zero entries with one entry in each jump vector indicate a transition **into** or **out of** the system.

Throughput of Arrivals

Consider the cases:

- $\{-1\} \in \mu_i^l$ and $\{+1\} \in \overline{\mu}_i^l$: this represents a departure from the system
- $\{+1\} \in \mu_i^l$ and $\{-1\} \in \overline{\mu}_i^l$: this represents an entry into the system

Throughput of Arrivals

Consider the cases:

- $\{-1\} \in \mu_i^l$ and $\{+1\} \in \bar{\mu}_i^l$: this represents a departure from the system
- $\{+1\} \in \mu_i^l$ and $\{-1\} \in \bar{\mu}_i^l$: this represents an entry into the system

Based on this reasoning we can define the throughput of arrivals:

Throughput of arrivals

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

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

Number in the System

Number in the System

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

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

Convergence and Average Response Time

Convergence

For any $S^i \in \mathcal{C}_i, S^i \neq \emptyset$, it holds that

$$\lambda_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} \lambda_{S^i}(x(t))$$

and that

$$L_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} L_{S^i}(x(t))$$

.

Convergence and Average Response Time

Convergence

For any $S^i \in \mathcal{C}_i, S^i \neq \emptyset$, it holds that

$$\lambda_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} \lambda_{S^i}(x(t))$$

and that

$$L_{S^i}(X_n(t)/n) \xrightarrow{\mathbb{E}} L_{S^i}(x(t))$$

.

Average Response Time

For any $S^i \in \mathcal{C}_i, S^i \neq \emptyset$, W_{S^i} is

$$W_{S^i}(\omega) = L_{S^i}(\omega) / \lambda_{S^i}(\omega)$$

Average response time for the example

$$\text{Download} \stackrel{\text{def}}{=} (\text{transfer}, r_1). \text{Think}$$

$$\text{Think} \stackrel{\text{def}}{=} (\text{think}, r_2). \text{Download}$$

$$\text{Upload} \stackrel{\text{def}}{=} (\text{transfer}, r_3). \text{Log}$$

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

$$\text{System} \stackrel{\text{def}}{=} \text{Download}[N_C] \underset{\{\text{transfer}\}}{\boxtimes} \text{Upload}[N_S]$$

If we define the partition to be $S^i = \{\text{Download}\}$, $\overline{S^i} = \{\text{Think}\}$, then

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

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

$$W_{S^i}(\omega) = \frac{\omega_1}{r_2 \omega_2}$$

Fluid Rewards in detail

In these slides I have given an overview of the types of measures that can be automatically derived.

Fluid Rewards in detail

In these slides I have given an overview of the types of measures that can be automatically derived.

As you have already seen these are implemented in the PEPA Eclipse plug-in tool under the **Scalable Analysis** in the **PEPA** menu.

Fluid Rewards in detail

In these slides I have given an overview of the types of measures that can be automatically derived.

As you have already seen these are implemented in the PEPA Eclipse plug-in tool under the **Scalable Analysis** in the **PEPA** menu.

Full details can be found in the paper:

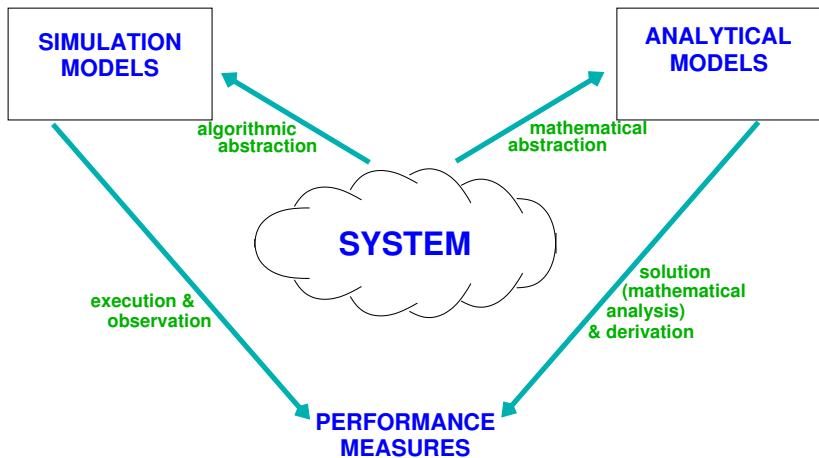


M. Tribastone, J. Ding, S. Gilmore, J. Hillston
Fluid Rewards for a Stochastic Process Algebra
IEEE Trans. Software Eng. 38(4): 861-874 (2012)

Outline

- 1 Fluid Rewards
- 2 Introduction to Simulation**
- 3 Simulation in PEPA

Introduction



Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.

Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the “environment” (represented by random variables within the model) the stochastic process progresses from state to state.

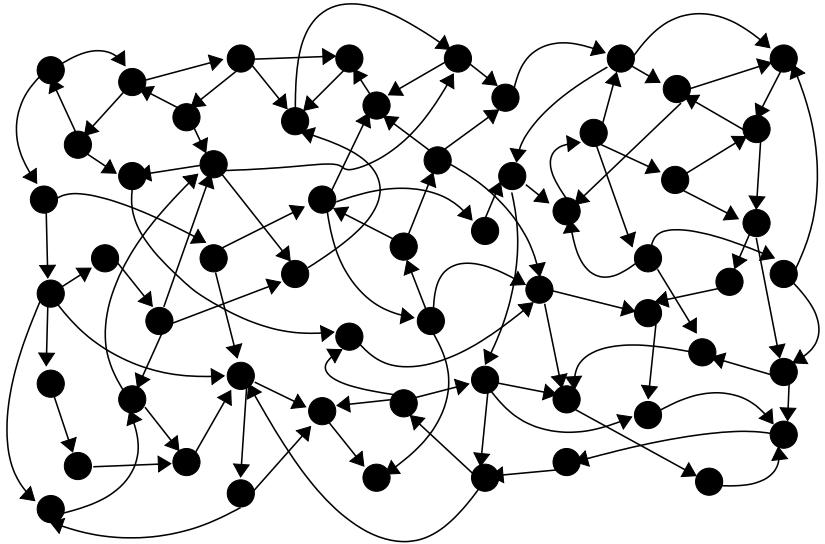
Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the “environment” (represented by random variables within the model) the stochastic process progresses from state to state.
- Any set of instances of $\{X(t), t \in T\}$ can be regarded as a path of a particle moving randomly in a state space, S , its position at time t being $X(t)$.

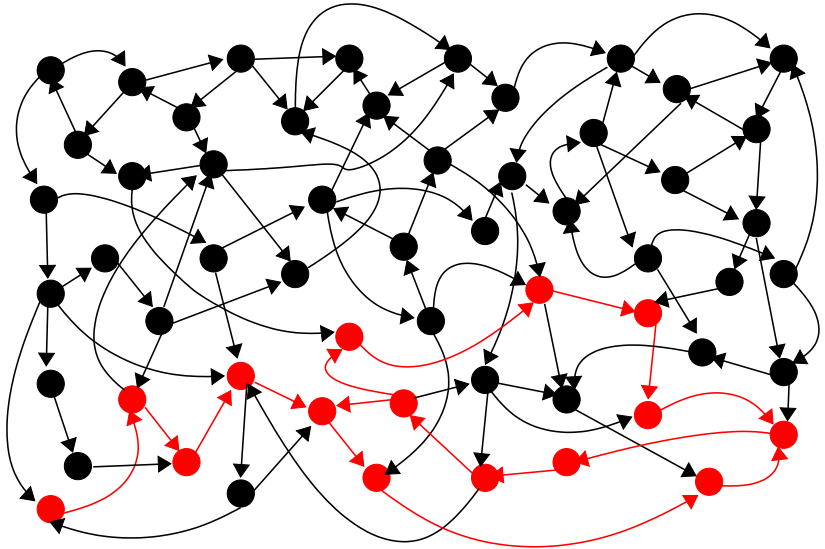
Assumptions

- We still assume that the system is characterised by a family of random variables $\{X(t), t \in T\}$.
- As the value of time increases, and in response to the “environment” (represented by random variables within the model) the stochastic process progresses from state to state.
- Any set of instances of $\{X(t), t \in T\}$ can be regarded as a path of a particle moving randomly in a state space, S , its position at time t being $X(t)$.
- These paths are called **sample paths**.

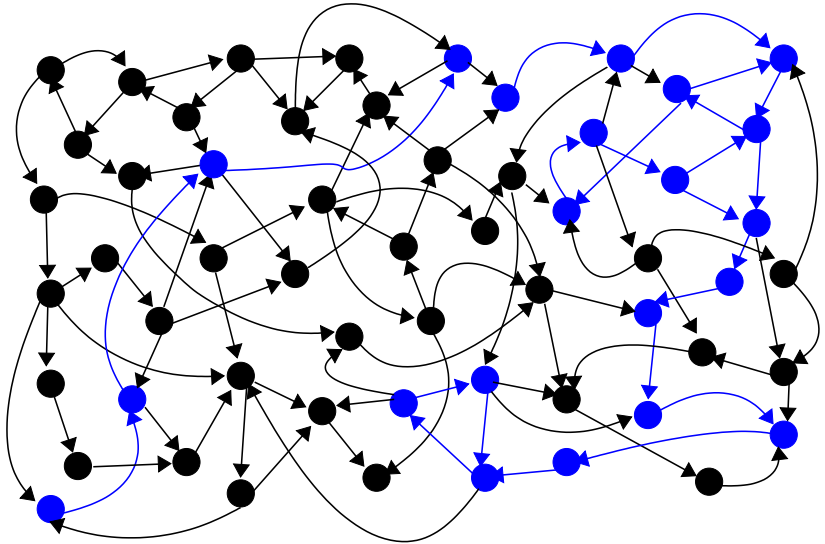
State space and sample paths



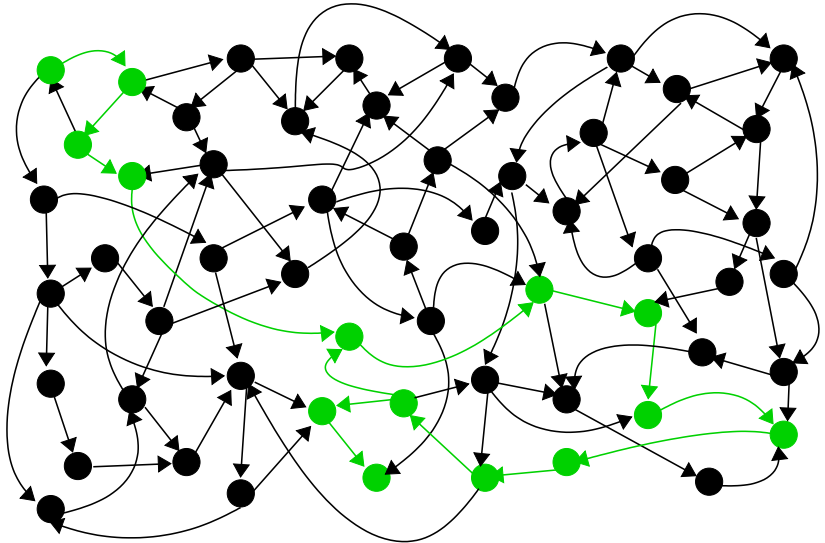
State space and sample paths



State space and sample paths



State space and sample paths



Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.

Sample paths and runs

- Using the analytic approach of Markov processes we characterised **all possible sample paths** by the global balance equations.
- Using simulation we investigate the sample paths directly.
- We allow the model to trace out a sample path over the state space.
- Each **run** of the simulation model will generate another, usually distinct, sample path.

Benefits of simulation

There are a variety of reasons in general why simulation may be preferable to analytical modelling:

- Level of Abstraction** It is not necessary to adhere to the assumptions of Markovian modelling (although we will in the simulation of PEPA models).
- Transient Analysis** As we have seen, transient analysis is possible via numerical solution of a CTMC but it is computationally costly, and can be easier to conduct via simulation.
- Size of State Space** In contrast to numerical solution of a CTMC, in a simulation model the state space is generated “on-the-fly” by the model itself during execution so it does not need to be all stored at once.

Simulation management

Some of the common features of simulation management are listed below.

- Event scheduler
- Simulation clock and time management
- System state variables
- Event routines
- Random number/random variate generator
- Report generator
- Trace routines
- Dynamic memory management

Event scheduler

An event scheduler keeps track of the events which are waiting to happen, usually as a linked list, and allows them to be manipulated in various ways. For example,

- schedule event E at time T ;
- hold event E for a time interval ∂t ;
- cancel a previously scheduled event E ;
- hold event E indefinitely (until it is scheduled by another event);
- schedule an indefinitely held event.

Event scheduler

An event scheduler keeps track of the events which are waiting to happen, usually as a linked list, and allows them to be manipulated in various ways. For example,

- schedule event E at time T ;
- hold event E for a time interval ∂t ;
- cancel a previously scheduled event E ;
- hold event E indefinitely (until it is scheduled by another event);
- schedule an indefinitely held event.

Event scheduler must be efficient

The event scheduler is called before every event, and it may be called several times during one event to schedule other new events.

Simulation clock and time management

- Every simulation model must have a global variable representing the **simulated** time.
- The event scheduler is usually responsible for advancing this time, either one unit at a time or, more commonly, directly to the time of the next scheduled event.
- This latter approach is called **event-driven** time management.

Event routines

- Each **event** in the system brings about a state change.
- In the simulation model the effect of each event must be represented in a way which updates the system state variables, and in some cases, schedules other events.
- How the event routines are generated will depend on the simulation modelling paradigm used to construct the model.

Random number/random variate generator

- Random numbers play a crucial role in most discrete event simulations.
- A random number generator is used to generate a sequence of random values between 0 and 1.
- These values are then transformed to produce a sequence of random values which satisfy the desired distribution. This second step is sometimes called **random variate generation**.

Random number/random variate generator

- Random numbers play a crucial role in most discrete event simulations.
- A random number generator is used to generate a sequence of random values between 0 and 1.
- These values are then transformed to produce a sequence of random values which satisfy the desired distribution. This second step is sometimes called **random variate generation**.

Example

The impact of the environment on the system, e.g. inter-arrival times, is usually represented by random variables of some specified distribution.

Simulation output analysis

- In performance modelling our objective in constructing a simulation model of a system is to generate one or more **performance measures** for the system.

Simulation output analysis

- In performance modelling our objective in constructing a simulation model of a system is to generate one or more **performance measures** for the system.
- In the Markov models such measures were **derived from the steady state probability distribution**, after the model solution.

Simulation output analysis

- In performance modelling our objective in constructing a simulation model of a system is to generate one or more **performance measures** for the system.
- In the Markov models such measures were **derived from the steady state probability distribution**, after the model solution.
- In contrast, in a simulation model measures are **observed or evaluated directly during the execution** of the model.

Simulation output analysis

- In performance modelling our objective in constructing a simulation model of a system is to generate one or more **performance measures** for the system.
- In the Markov models such measures were **derived from the steady state probability distribution**, after the model solution.
- In contrast, in a simulation model measures are **observed or evaluated directly during the execution** of the model.
- It is part of model construction to make sure that all the necessary counters and updates are in place to allow the measures to be collected as the model runs.

Simulation trajectories

It is important to remember that each run of a model constitutes a **single trajectory** over the state space.

Simulation trajectories

It is important to remember that each run of a model constitutes a **single trajectory** over the state space.

So, in general, any estimate for the value of a performance measure generated from a single run constitutes a **single observation** in the possible sample space.

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on a single observation.

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on a **single observation**.
- For **steady state analysis** the averages we calculate from data collected during execution will always be an approximation of the unknown true long-term averages that characterise the system performance.

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on a **single observation**.
- For **steady state analysis** the averages we calculate from data collected during execution will always be an approximation of the unknown true long-term averages that characterise the system performance.
- Important issues are:

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on a **single observation**.
- For **steady state analysis** the averages we calculate from data collected during execution will always be an approximation of the unknown true long-term averages that characterise the system performance.
- Important issues are:
 - choosing the **starting state of the simulation**;

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on a **single observation**.
- For **steady state analysis** the averages we calculate from data collected during execution will always be an approximation of the unknown true long-term averages that characterise the system performance.
- Important issues are:
 - choosing the **starting state of the simulation**;
 - choosing the **warm-up** period that is allowed to elapse before data collection begins;

Simulation and long-term averages

- To gain an accurate measure of the performance of the system we should not base our results on **a single observation**.
- For **steady state analysis** the averages we calculate from data collected during execution will always be an approximation of the unknown true long-term averages that characterise the system performance.
- Important issues are:
 - choosing the **starting state of the simulation**;
 - choosing the **warm-up** period that is allowed to elapse before data collection begins;
 - choosing a **run length** that ensures that the calculated averages are representative of the unknown true long term average.

Statistical techniques

- Statistical techniques can be used to assess how and when the calculated averages approximate the true average, i.e. to analyse the accuracy of our current estimate.
- This is often done in terms of a **confidence interval**.
- A confidence interval expresses probabilistic bounds on the error of our current estimate.

Confidence intervals

A confidence interval (c_1, c_2) with **confidence level** $X\%$, means that with probability $X/100$ the real value v lies between the values c_1 and c_2 , i.e.

$$\Pr(c_1 \leq v \leq c_2) = X/100$$

Confidence intervals

A confidence interval (c_1, c_2) with **confidence level** $X\%$, means that with probability $X/100$ the real value v lies between the values c_1 and c_2 , i.e.

$$\Pr(c_1 \leq v \leq c_2) = X/100$$

$X/100$ is usually written in the form $1 - \alpha$, and α is called the **significance level**, and $(1 - \alpha)$ is called the **confidence coefficient**.

Confidence intervals and variance

Usually performance modellers will run their simulation models until their observations give them confidence levels of 90% or 95% and a confidence interval which is acceptably tight.

Confidence intervals and variance

Usually performance modellers will run their simulation models until their observations give them confidence levels of 90% or 95% and a confidence interval which is acceptably tight.

Calculation of the confidence interval is based on the **variance** within the observations which have been gathered.

Confidence intervals and variance

Usually performance modellers will run their simulation models until their observations give them confidence levels of 90% or 95% and a confidence interval which is acceptably tight.

Calculation of the confidence interval is based on the **variance** within the observations which have been gathered.

The greater the variance, the wider the confidence interval; the smaller the variance, the tighter the bounds.

Confidence intervals with PEPA

In the PEPA Eclipse Plug-in it is possible to set the desired confidence interval as a stopping criterion for a simulation run.

Confidence intervals with PEPA

In the PEPA Eclipse Plug-in it is possible to set the desired confidence interval as a stopping criterion for a simulation run.

Clearly, the tighter the confidence intervals the more runs are likely to be needed to achieve it.

Confidence intervals with PEPA

In the PEPA Eclipse Plug-in it is possible to set the desired confidence interval as a stopping criterion for a simulation run.

Clearly, the tighter the confidence intervals the more runs are likely to be needed to achieve it.

But note that the larger the populations that you are simulating the easier it will be to get a tighter confidence interval.

Confidence intervals with PEPA

In the PEPA Eclipse Plug-in it is possible to set the desired confidence interval as a stopping criterion for a simulation run.

Clearly, the tighter the confidence intervals the more runs are likely to be needed to achieve it.

But note that the larger the populations that you are simulating the easier it will be to get a tighter confidence interval.

However, one way or the other it is computationally expensive to get tight confidence intervals.

Initial conditions, bias

The initial conditions of the model, its **starting state**, influence the sequence of states through which the simulation will pass, especially near the start of a run.

Initial conditions, bias

The initial conditions of the model, its **starting state**, influence the sequence of states through which the simulation will pass, especially near the start of a run.

In a steady state distribution the output values should be **independent of the starting state**.

Initial conditions, bias

The initial conditions of the model, its **starting state**, influence the sequence of states through which the simulation will pass, especially near the start of a run.

In a steady state distribution the output values should be **independent of the starting state**.

Thus the modeller must make some effort to remove the effect of the starting state, sometimes termed **bias**, from the sample data used for estimating the performance measure of interest.

Initial conditions, bias

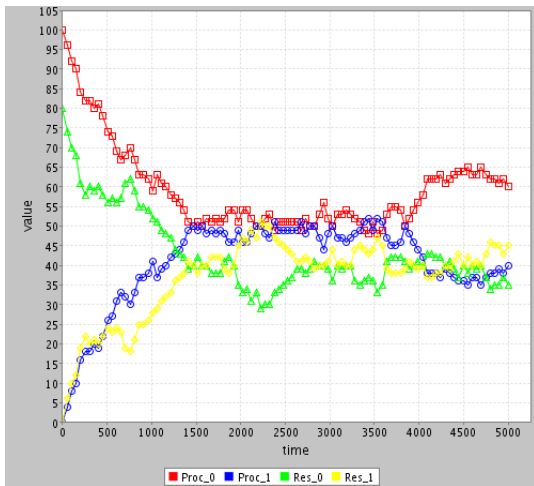
The initial conditions of the model, its **starting state**, influence the sequence of states through which the simulation will pass, especially near the start of a run.

In a steady state distribution the output values should be **independent of the starting state**.

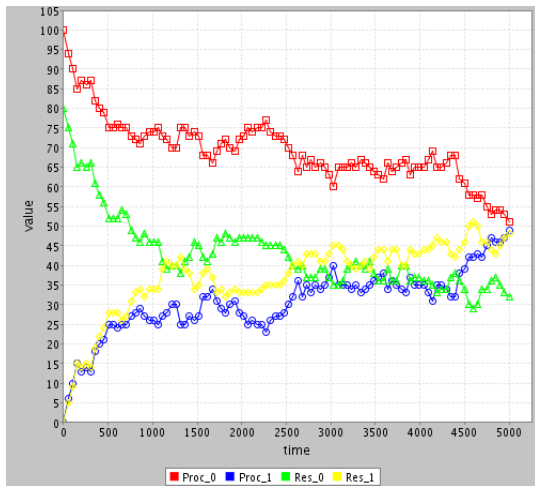
Thus the modeller must make some effort to remove the effect of the starting state, sometimes termed **bias**, from the sample data used for estimating the performance measure of interest.

Unfortunately it is not possible to define exactly when the model has moved from transient behaviour to steady state behaviour. This initial period before steady state is reached is sometimes called the **warm-up period**.

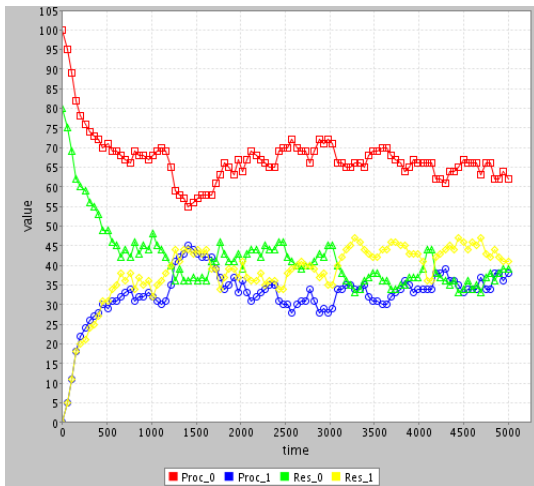
100 processors and 80 resources (simulation run A)



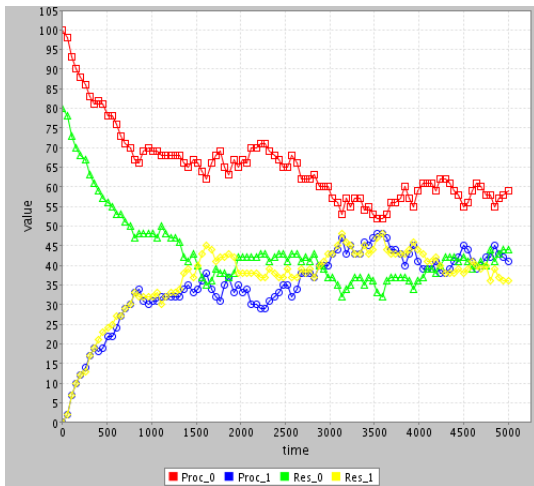
100 processors and 80 resources (simulation run B)



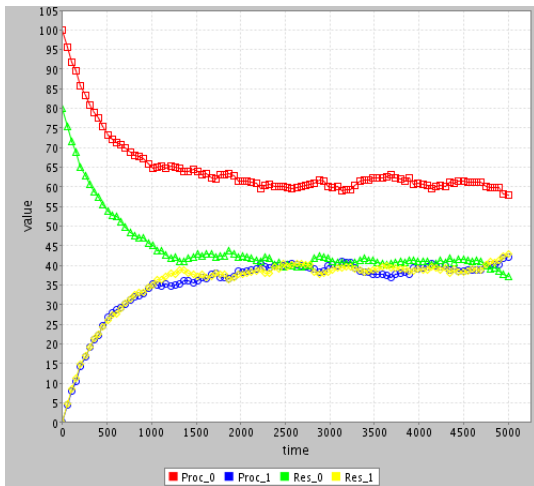
100 processors and 80 resources (simulation run C)



100 processors and 80 resources (simulation run D)



100 processors and 80 resources (average of 10 runs)



Heuristics for reducing bias

The common techniques are

- 1 Long runs.
- 2 Proper initialisation.
- 3 Truncation.
- 4 Initial data deletion.
- 5 Moving average of independent replications.
- 6 Batch means.

Heuristics for reducing bias

The common techniques are

- 1 Long runs.
- 2 Proper initialisation.
- 3 Truncation.
- 4 Initial data deletion.
- 5 Moving average of independent replications.
- 6 Batch means.

The last four techniques are all based on the assumption that **variability is less during steady state** behaviour than during transient behaviour.

Variance reduction techniques

- Assume that we are running a simulation model in order to estimate some performance measure M .
- During the i th execution of the model we make observations of M , o_{ij} and at the end of the run we calculate the mean value of the observations O_i .
- Note that the observations o_{ij} in most simulations are **not** independent. Successive observations are often correlated.

Example of correlation

- If we are interested in the delay of messages in a packet-switching network, if the delay of one message is long because the network is heavily congested, the next message is likely to be similarly delayed.

Example of correlation

- If we are interested in the delay of messages in a packet-switching network, if the delay of one message is long because the network is heavily congested, the next message is likely to be similarly delayed.
- Thus the two observations are not independent.

Example of correlation

- If we are interested in the delay of messages in a packet-switching network, if the delay of one message is long because the network is heavily congested, the next message is likely to be similarly delayed.
- Thus the two observations are not independent.

Note

This is why, in general, a simulation model must be run several times.

Independent replications

- If independent replications are used the model is run m times in order to generate m independent observations.

Independent replications

- If independent replications are used the model is run m times in order to generate m independent observations.
- For the runs to be independent, the random number generator seeds must be carefully chosen to ensure that they are independent.

Independent replications

- If independent replications are used the model is run m times in order to generate m independent observations.
- For the runs to be independent, the random number generator seeds must be carefully chosen to ensure that they are independent.
- If steady state or long term behaviour is being investigated the data relating to the warm-up period must be discarded.

Independent replications

- If independent replications are used the model is run m times in order to generate m independent observations.
- For the runs to be independent, the random number generator seeds must be carefully chosen to ensure that they are independent.
- If steady state or long term behaviour is being investigated the data relating to the warm-up period must be discarded.
- Let O denote the mean value of the retained observations, O_i , after m runs.

Independent replications

- If independent replications are used the model is run m times in order to generate m independent observations.
- For the runs to be independent, the random number generator seeds must be carefully chosen to ensure that they are independent.
- If steady state or long term behaviour is being investigated the data relating to the warm-up period must be discarded.
- Let O denote the mean value of the retained observations, O_i , after m runs.
- The variance over all observations is calculated as shown below:

$$V = \frac{1}{m-1} \sum_{i=1}^m (O_i - O)^2$$

Independent replications and steady-state

For steady-state analysis independent replication is an inefficient way to generate samples, since for each sample point, O_i , k observations, $\{o_{i1}, \dots, o_{ik}\}$, must be discarded.

Batch means

- In the method of batch means the model is run only once but for a very long period.

Batch means

- In the method of batch means the model is run only once but for a very long period.
- The run is divided into a series of sub-periods of length ℓ , and measures are collected over each sub-run to form a single point estimate.

Batch means

- In the method of batch means the model is run only once but for a very long period.
- The run is divided into a series of sub-periods of length ℓ , and measures are collected over each sub-run to form a single point estimate.
- If the observations made during the run form a set $\{o_i\}$, the set is partitioned into subsets

$$S_i = \{o_j \mid o_j \text{ observed between } (i - 1) \times \ell \text{ and } i \times \ell\}$$

Batch means

- In the method of batch means the model is run only once but for a very long period.
- The run is divided into a series of sub-periods of length ℓ , and measures are collected over each sub-run to form a single point estimate.
- If the observations made during the run form a set $\{o_i\}$, the set is partitioned into subsets

$$S_i = \{o_j \mid o_j \text{ observed between } (i-1) \times \ell \text{ and } i \times \ell\}$$

- Now each sample point O_i is the mean generated from a subset of observations S_i , and O is the mean generated from the O_i .

Batch means

- In the method of batch means the model is run only once but for a very long period.
- The run is divided into a series of sub-periods of length ℓ , and measures are collected over each sub-run to form a single point estimate.
- If the observations made during the run form a set $\{o_i\}$, the set is partitioned into subsets

$$S_i = \{o_j \mid o_j \text{ observed between } (i-1) \times \ell \text{ and } i \times \ell\}$$

- Now each sample point O_i is the mean generated from a subset of observations S_i , and O is the mean generated from the O_i .
- Variance is calculated as above.

Batch means and independence

This method is unreliable since the sub-periods are clearly not independent.

Batch means and independence

This method is unreliable since the sub-periods are clearly not independent.

However it has the advantage that only one set of observations $\{o_i \dots o_k\}$ needs to be discarded to overcome the warm-up effects in steady state analysis.

Outline

- 1 Fluid Rewards
- 2 Introduction to Simulation
- 3 Simulation in PEPA**

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead **finding the states step-by-step** as the simulation progresses.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead **finding the states step-by-step** as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead **finding the states step-by-step** as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

This means that we do not need to maintain an **event list**.

Simulation in PEPA

When we simulate PEPA models we are simulating the underlying Markov process, avoiding the construction of the whole state space at once, instead **finding the states step-by-step** as the simulation progresses.

Because we are working in the Markovian context we can take advantage of the memoryless property.

This means that we do not need to maintain an **event list**.

In this case the simulation algorithm is particularly simple and relatively efficient.

The Gillespie Stochastic Simulation Algorithm

Instead of an event list the simulation engine keeps the **state of the system** and so knows for each component what activity or activities it currently enables (for shared activities it will check that all participating components are able to undertake the actions).

The Gillespie Stochastic Simulation Algorithm

Instead of an event list the simulation engine keeps the **state of the system** and so knows for each component what activity or activities it currently enables (for shared activities it will check that all participating components are able to undertake the actions).

From this list of **possible activities** it will select one to execute according to the **race policy** and then update the state accordingly, modifying the list of current activities as necessary.

Two Observations

If we have a number of possible activities $(\alpha_1, r_1), (\alpha_2, r_2), \dots, (\alpha_n, r_n)$ enabled in the current state, then we know from the superposition principle for the exponential distribution that the time until **something** happens is governed by an exponential distribution with **rate** $r_1 + r_2 + \dots + r_n$.

Two Observations

If we have a number of possible activities $(\alpha_1, r_1), (\alpha_2, r_2), \dots, (\alpha_n, r_n)$ enabled in the current state, then we know from the superposition principle for the exponential distribution that the time until **something** happens is governed by an exponential distribution with **rate** $r_1 + r_2 + \dots + r_n$.

We also know that the probability that the activity of type α_i is the one which will win the rate is

$$\frac{r_i}{r_1 + r_2 + \dots + r_n}.$$

The Gillespie Stochastic Simulation Algorithm

Thus we need only draw two random numbers for each step of the simulation algorithm:

The Gillespie Stochastic Simulation Algorithm

Thus we need only draw two random numbers for each step of the simulation algorithm:

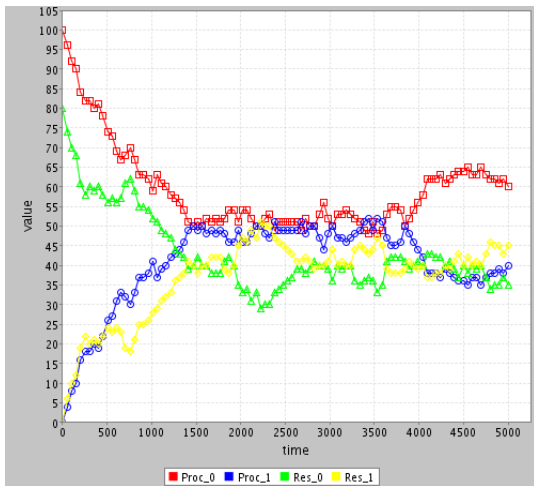
- the first determines the delay until the next activity completes,

The Gillespie Stochastic Simulation Algorithm

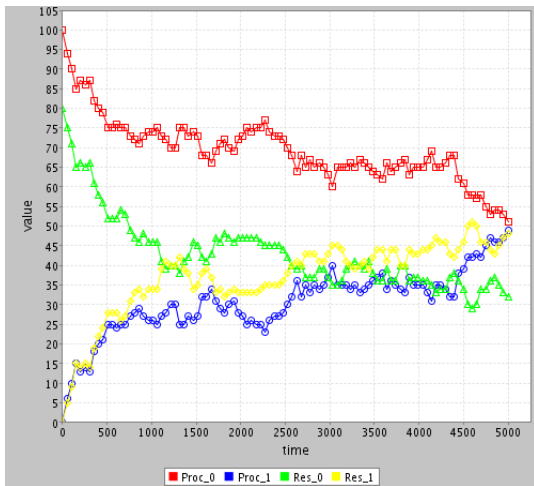
Thus we need only draw two random numbers for each step of the simulation algorithm:

- the first determines the delay until the next activity completes,
- the second determines which activity that will be.

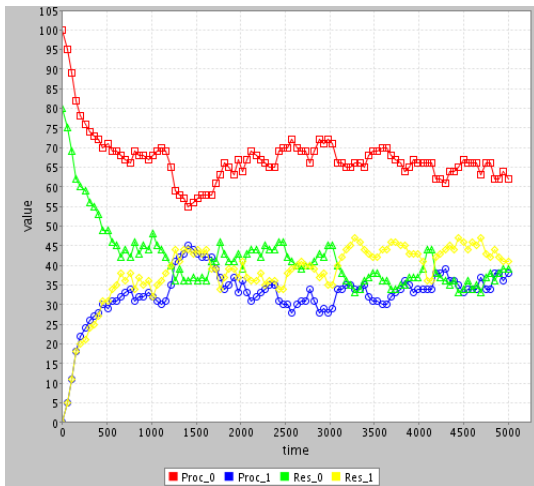
100 processors and 80 resources (simulation run A)



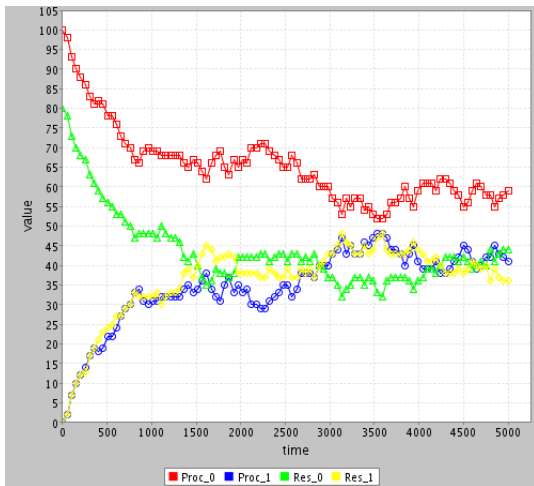
100 processors and 80 resources (simulation run B)



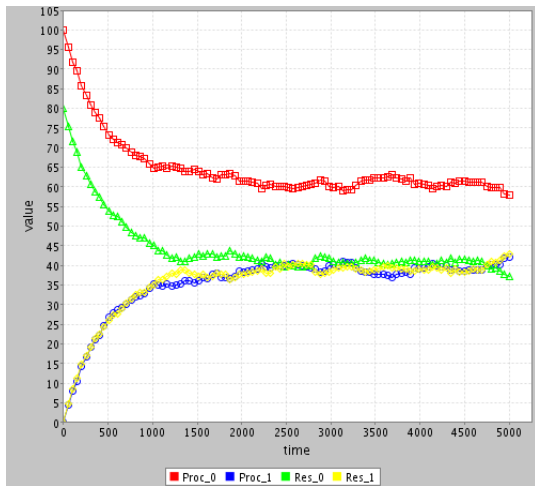
100 processors and 80 resources (simulation run C)



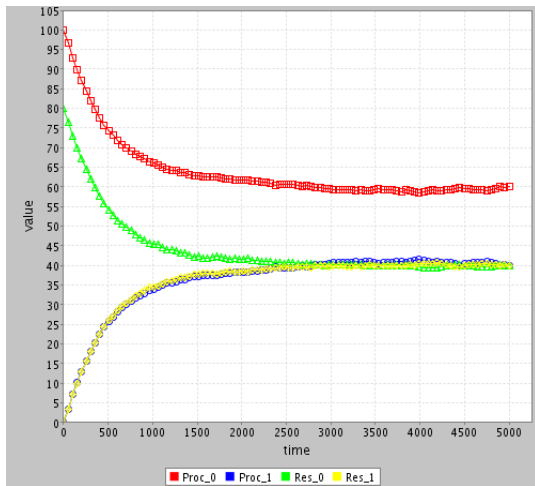
100 processors and 80 resources (simulation run D)



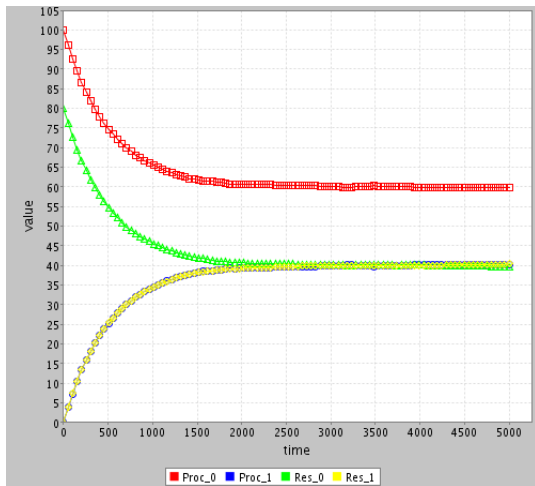
100 processors and 80 resources (average of 10 runs)



100 Processors and 80 resources (average of 100 runs)



100 processors and 80 resources (average of 1000 runs)



Differential Analysis View

- Extracting generating functions from a PEPA model presents little computational challenge because it does not require the exploration of the whole state space of the CTMC.

Differential Analysis View

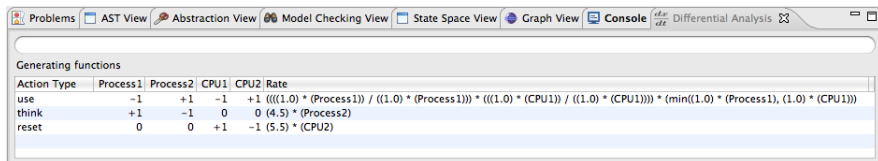
- Extracting generating functions from a PEPA model presents little computational challenge because it does not require the exploration of the whole state space of the CTMC.
- In the plug-in, the **Differential Analysis View** updates the generating functions of the currently active model whenever its contents are saved.

Differential Analysis View

- Extracting generating functions from a PEPA model presents little computational challenge because it does not require the exploration of the whole state space of the CTMC.
- In the plug-in, the **Differential Analysis View** updates the generating functions of the currently active model whenever its contents are saved.

Differential Analysis View

- Extracting generating functions from a PEPA model presents little computational challenge because it does not require the exploration of the whole state space of the CTMC.
- In the plug-in, the **Differential Analysis View** updates the generating functions of the currently active model whenever its contents are saved.



Generating functions

Action Type	Process1	Process2	CPU1	CPU2	Rate
use	-1	+1	-1	+1	$((((1.0) * (\text{Process1})) / ((1.0) * (\text{Process1}))) * (((1.0) * (\text{CPU1})) / ((1.0) * (\text{CPU1})))) * (\min((1.0) * (\text{Process1}), (1.0) * (\text{CPU1})))$
think	+1	-1	0	0	$(4.5) * (\text{Process2})$
reset	0	0	+1	-1	$(5.5) * (\text{CPU2})$

Differential Analysis View

- Extracting generating functions from a PEPA model presents little computational challenge because it does not require the exploration of the whole state space of the CTMC.
- In the plug-in, the **Differential Analysis View** updates the generating functions of the currently active model whenever its contents are saved.

Action Type	Process1	Process2	CPU1	CPU2	Rate
use	-1	+1	-1	+1	$\frac{(((1.0) * (\text{Process1})) / ((1.0) * (\text{Process1}))) * (((1.0) * (\text{CPU1})) / ((1.0) * (\text{CPU1})))) * (\min(1.0) * (\text{Process1}), (1.0) * (\text{CPU1}))}$
think	+1	-1	0	0	$(4.5) * (\text{Process2})$
reset	0	0	+1	-1	$(5.5) * (\text{CPU2})$

Generating functions

Action Type	Process1	Process2	CPU1	CPU2	Rate
use	-1	+1	-1	+1	$\frac{(((1.0) * (\text{Proce$
think	+1	-1	0	0	$(4.5) * (\text{Process$
reset	0	0	+1	-1	$(5.5) * (\text{CPU2})$

Stochastic simulation

- The generating functions contain all the necessary information for model analysis and admit a straightforward stochastic simulation algorithm.

Stochastic simulation

- The generating functions contain all the necessary information for model analysis and admit a straightforward stochastic simulation algorithm.
- Given a state $\hat{\xi}$, the evaluation of each of the generating functions $f_\alpha(\hat{\xi}, l)$ of the model gives the relative probabilities with which each action may be performed.

Stochastic simulation

- The generating functions contain all the necessary information for model analysis and admit a straightforward stochastic simulation algorithm.
- Given a state $\hat{\xi}$, the evaluation of each of the generating functions $f_\alpha(\hat{\xi}, l)$ of the model gives the relative probabilities with which each action may be performed.
- Drawing a random number from the resulting probability density function decides which action is to be taken, and thus the corresponding target state $\hat{\xi} + l$.

Stochastic simulation

- The generating functions contain all the necessary information for model analysis and admit a straightforward stochastic simulation algorithm.
- Given a state $\hat{\xi}$, the evaluation of each of the generating functions $f_{\alpha}(\hat{\xi}, l)$ of the model gives the relative probabilities with which each action may be performed.
- Drawing a random number from the resulting probability density function decides which action is to be taken, and thus the corresponding target state $\hat{\xi} + l$.
- This procedure may be repeated until conditions of termination of the simulation algorithm are met.

Stochastic simulation dialogue

Population Level Analysis

Population Level Analysis

Kind of analysis	Transient
Convergence criterion	Confidence level
Start time	0.0
Stop time	5.0
Number of time points	100
Number of replications	100
Confidence level	0.95
Confidence level percentage error	1.0

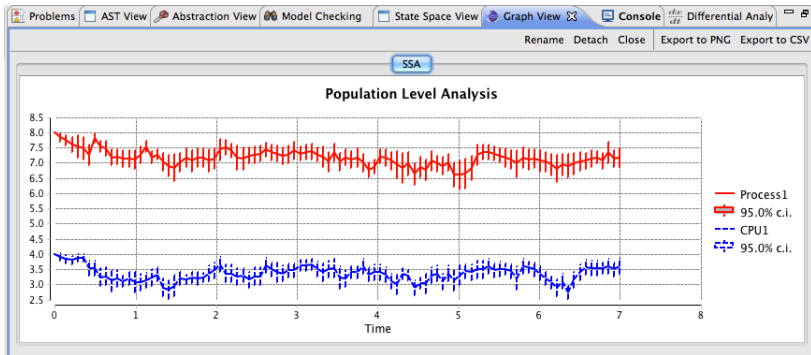
Select population levels

Search

- Process1
- Process2
- CPU1
- CPU2

? Analyse Cancel

Results of a transient stochastic simulation



Simulation methods

- Transient simulation is based on the method of **independent replications**: steady-state simulation is performed with the method of **batch means**.
- At the end of a batch, the algorithm checks whether the tracked population counts have reached the desired confidence level.
- If the maximum number of batches is reached, the algorithm returns with a warning of potentially bad accuracy.
- The lag-1 correlation is also computed as an indicator of statistical independence between adjacent batches.

Steady-state results

Simulation results



Runtime: 156 ms.

Results:

Measure: Process1

Average: 7.252427

95.00% Confidence Interval: 0.102%

Lag-1 Correlation: 1.401892e-05

Measure: Process2

Average: 0.747573

95.00% Confidence Interval: 0.990%

Lag-1 Correlation: 1.134021e-03

Measure: CPU1

Average: 3.386645

95.00% Confidence Interval: 0.152%

Lag-1 Correlation: 1.330295e-05

Measure: CPU2

Average: 0.613355

95.00% Confidence Interval: 0.838%

Lag-1 Correlation: 3.965669e-04

ODE results



Runtime: 43ms.

Process1 : 7.247863

Process2 : 0.752137

CPU1 : 3.384615

CPU2 : 0.615385

Convergence norm is: 6.349282e-07

Steady state detected at 3.393 time units