The time/space continuum: Continuous-time and continuous-space process algebras

Stephen Gilmore

LFCS, University of Edinburgh

PASTA Workshop, Edinburgh, 7th September 2005

Background

This talk is about PEPA.

Background

This talk is about PEPA.

PEPA with CMTC semantics — a continuous-time process algebra
PEPA with ODE semantics — a continuous-space process algebra

Background

This talk is about PEPA.

PEPA with CMTC semantics — a continuous-time process algebra
PEPA with ODE semantics — a continuous-space process algebra

Are they different?

Background: Deterministic processes

A process is a deterministic process if knowledge of its values up to and including time t allows us to unambiguously predict its value at any infinitesimally later time $t + dt$.

Background: ODEs are memoryless deterministic processes

A set of ordinary differential equations defines a memoryless deterministic process.

$$\begin{aligned} \mathbf{X}(t + dt) &= \mathbf{X}(t) + f(\mathbf{X}(t), t)dt \\ \frac{d\mathbf{X}}{dt} &= f(\mathbf{X}, t) \end{aligned}$$

Background: Stochastic processes

A process is a stochastic process if knowledge of its values up to and including time t allows us to probabilistically predict its value at any infinitesimally later time $t + dt$.

Background: Stochastic processes

A process is a stochastic process if knowledge of its values up to and including time t allows us to probabilistically predict its value at any infinitesimally later time $t + dt$.

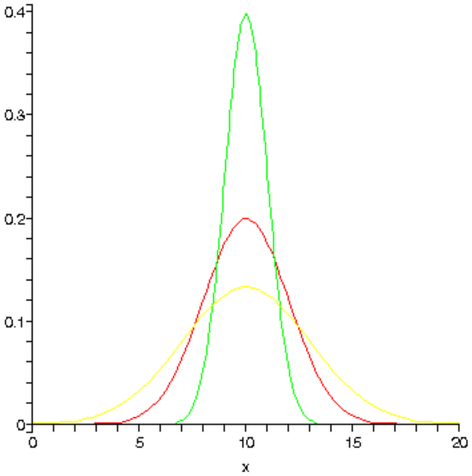
Stochastic processes subsume deterministic processes.

Background: CTMCs are memoryless stochastic processes

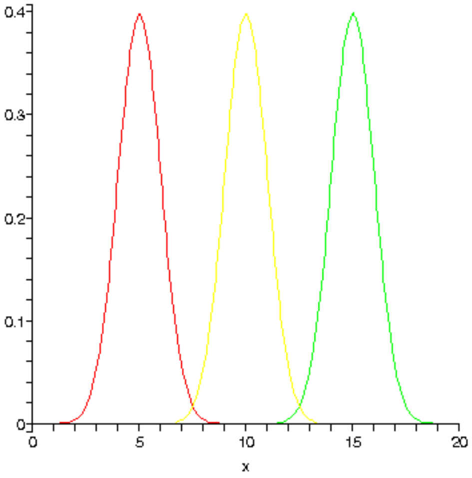
A continuous-time Markov chain is a memoryless stochastic process.

$$
\begin{aligned}
& \Pr(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n, \ldots, X(t_1) = x_1) \\
= \quad & \Pr(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n)
\end{aligned}
$$

Background: Same mean, different standard deviations



Background: Same standard deviations, different mean



Background: converting PEPA to ODEs

Two classes of PEPA models can be used to generate ODEs.

High/Low models[1]: High and low concentrations of components are modelled, to indicate increase or decrease in quantity.

Direct style[2]: Models encode the behaviour of the system directly without the use of high and low labels.

This talk: models in direct style.

---

[1] "Automatically deriving ODEs from process algebra models of signalling pathways", Muffy Calder, Stephen Gilmore and Jane Hillston, Computational Methods in Systems Biology (CMSB 2005), Edinburgh, Scotland, April 2005.

[2] "Fluid Flow Approximation of PEPA models", Jane Hillston, Quantitative Evaluation of SysTems (QEST 2005), Torino, Italy, September 2005.

Outline

1. Quantitative modelling with CTMCs and ODEs
   - Modelling with quantified process algebras
   - Analysis based on Continuous-time Markov Chains
   - Analysis based on Ordinary Differential Equations
2. Performance modelling with process algebras
   - Performance Evaluation Process Algebra
   - PEPA model of jobs and servers
   - Analysis of the model
3. Comparing performance measures
   - Computed with continuous time
   - Computed with continuous space
   - Comparison of computed measures
4. Commentary and comparison

Outline

Modelling with quantified process algebras

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Modelling with quantified process algebras

Tiny example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$

This example defines a system with nine reachable states:

1. $P_1 \parallel P_1$
2. $P_1 \parallel P_2$
3. $P_1 \parallel P_3$
4. $P_2 \parallel P_1$
5. $P_2 \parallel P_2$
6. $P_2 \parallel P_3$
7. $P_3 \parallel P_1$
8. $P_3 \parallel P_2$
9. $P_3 \parallel P_3$

The transitions between states have quantified duration r which can be evaluated against a CTMC or ODE interpretation.

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 0$:

1. 1.0000
2. 0.0000
3. 0.0000
4. 0.0000
5. 0.0000
6. 0.0000
7. 0.0000
8. 0.0000
9. 0.0000

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 1$:

1. 0.1642
2. 0.1567
3. 0.0842
4. 0.1567
5. 0.1496
6. 0.0804
7. 0.0842
8. 0.0804
9. 0.0432

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 2$:

1. 0.1056
2. 0.1159
3. 0.1034
4. 0.1159
5. 0.1272
6. 0.1135
7. 0.1034
8. 0.1135
9. 0.1012

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 3$:

1. 0.1082
2. 0.1106
3. 0.1100
4. 0.1106
5. 0.1132
6. 0.1125
7. 0.1100
8. 0.1125
9. 0.1119

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 4$:

1. 0.1106
2. 0.1108
3. 0.1111
4. 0.1108
5. 0.1110
6. 0.1113
7. 0.1111
8. 0.1113
9. 0.1116

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 5$:

1. 0.1111
2. 0.1110
3. 0.1111
4. 0.1110
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 6$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 7$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1111
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$
$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 0$:

P_1	2.0000
P_2	0.0000
P_3	0.0000

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 1$:

P_1	0.8121
P_2	0.7734
P_3	0.4144

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 2$:		
	P_1	0.6490
	P_2	0.7051
	P_3	0.6457

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 3$:

P_1	0.6587
P_2	0.6719
P_3	0.6692

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 4$:

P_1	0.6648
P_2	0.6665
P_3	0.6685

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 5$:

P_1	0.6666
P_2	0.6663
P_3	0.6669

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 6$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1)$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 7$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 0$:

P_1	3.0000
P_2	0.0000
P_3	0.0000

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 1$:

P_1	1.1782
P_2	1.1628
P_3	0.6590

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 2$:

P_1	0.9766
P_2	1.0754
P_3	0.9479

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 3$:

P_1	0.9838
P_2	1.0142
P_3	1.0020

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 4$:

P_1	0.9981
P_2	0.9995
P_3	1.0023

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 5$:		
	P_1	1.0001
	P_2	0.9996
	P_3	1.0003

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 6$:

P_1	1.0001
P_2	0.9999
P_3	1.0000

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$

$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 7$:

P_1	1.0000
P_2	0.9999
P_3	0.9999

Analysis based on Ordinary Differential Equations

Slightly larger example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 8$:

P_1	1.0000
P_2	1.0000
P_3	1.0000

What just happened?

An ODE specifies how the value of some continuous variable varies over continuous time. For example, the temperature in a container may be modelled by an ODE describing how the temperature will change dependent on the current temperature and pressure. The pressure can be similarly modelled and the equations together form a system of ODEs describing the state of the container.

What just happened?

In a PEPA model the state at any current time is the local derivative or state of each component of the model. When we have large numbers of repeated components it can make sense to represent each component type as a continuous variable, and the state of the model as a whole as the set of such variables. The evolution of each such variable can then be described by an ODE.

What just happened?

The PEPA definitions of the component specify the activities which can increase or decrease the number of components exhibited in the current state. The cooperations show when the number of instances of another component will have an influence on the evolution of this component.

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

That's not what we're doing. We go directly to ODEs.

What's the value proposition?

- The bottleneck for Markovian modelling of systems is the size of the solution vector, which is bounded by the product of the state-space sizes of the processes which are composed in parallel ("state-space explosion").

What's the value proposition?

- The bottleneck for Markovian modelling of systems is the size of the solution vector, which is bounded by the product of the state-space sizes of the processes which are composed in parallel ("state-space explosion").
- The size of the solution vector for the system of ODEs may be exponentially smaller.

Outline

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

The rate at which an activity is performed is quantified by some component in each co-operation. The symbol $\top$ indicates that the rate value is quantified elsewhere (not in this component).

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

The rate at which an activity is performed is quantified by some component in each co-operation. The symbol $\top$ indicates that the rate value is quantified elsewhere (not in this component).

$(\alpha, r).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
X	Variable

Derived forms and additional syntax

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Derived forms and additional syntax

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Because we are interested in transient behaviour we use the deadlocked process *Stop*.

Derived forms and additional syntax

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Because we are interested in transient behaviour we use the deadlocked process *Stop*.

When working with large numbers of jobs and servers, we write $P[n]$ to denote an array of n copies of P executing in parallel.

Derived forms and additional syntax

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Because we are interested in transient behaviour we use the deadlocked process *Stop*.

When working with large numbers of jobs and servers, we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

Modelling jobs and nodes

Consider jobs with a number of ordered stages. (Here three.)

Modelling jobs and nodes

Consider jobs with a number of ordered stages. (Here three.)

Jobs must be loaded onto a node before execution. Stage 1 must be completed before Stage 2 and Stage 2 before Stage 3. After Stage 3 the job is cleared by being unloaded from the node, and is then finished.

Modelling jobs and nodes

Consider jobs with a number of ordered stages. (Here three.)

Jobs must be loaded onto a node before execution. Stage 1 must be completed before Stage 2 and Stage 2 before Stage 3. After Stage 3 the job is cleared by being unloaded from the node, and is then finished.

Here the number of compute jobs is larger than the number of nodes available to execute them. Nodes specify the rate at which jobs are completed.

PEPA model of jobs and nodes

Jobs

$$
\begin{aligned}
Job &\stackrel{def}{=} (load, \top).Job1 \\
Job1 &\stackrel{def}{=} (stage1, \top).Job2 \\
Job2 &\stackrel{def}{=} (stage2, \top).Job3 \\
Job3 &\stackrel{def}{=} (stage3, \top).Clearing \\
Clearing &\stackrel{def}{=} (unload, \top).Finished \\
Finished &\stackrel{def}{=} Stop
\end{aligned}
$$

PEPA model of jobs and nodes

Nodes

$$
\begin{aligned}
\mathit{NodeIdle} &\stackrel{\mathit{def}}{=} (\mathit{load}, r_0).\mathit{Node1} \\
\mathit{Node1} &\stackrel{\mathit{def}}{=} (\mathit{stage1}, r_1).\mathit{Node2} \\
\mathit{Node2} &\stackrel{\mathit{def}}{=} (\mathit{stage2}, r_2).\mathit{Node3} \\
\mathit{Node3} &\stackrel{\mathit{def}}{=} (\mathit{stage3}, r_3).\mathit{Node4} \\
\mathit{Node4} &\stackrel{\mathit{def}}{=} (\mathit{unload}, r_0).\mathit{NodeIdle}
\end{aligned}
$$

PEPA model of jobs and nodes

System

$$\mathit{NodeIdle}[100] \underset{L}{\bowtie} \mathit{Job}[1000]$$

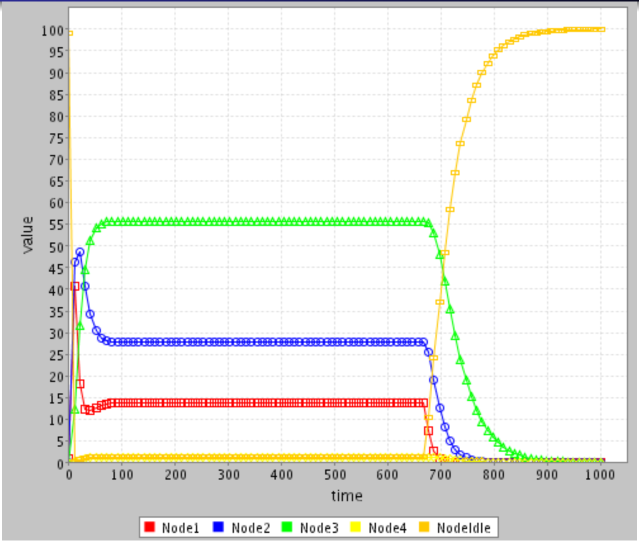
where L is $\{\, \mathit{load}, \mathit{stage1}, \mathit{stage2}, \mathit{stage3}, \mathit{unload} \,\}$.

Analysis of the model

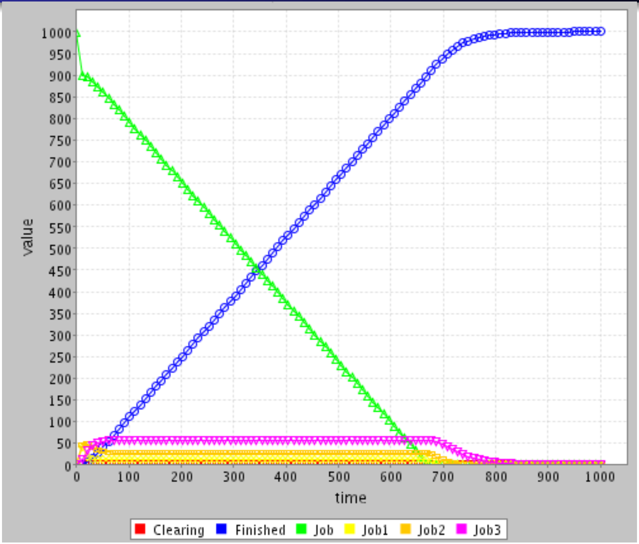
Analysis of the model proceeds by choosing particular values for the rates. The values below are chosen to make the analysis easy to follow.

Rate	Value	Interpretation
r_0	1	(Un)loading takes one time unit
r_1	0.1	Stage 1 takes ten time units
r_2	0.05	Stage 2 takes twenty time units
r_3	0.025	Stage 3 takes forty time units

Analysis of the model: Nodes



Analysis of the model: Jobs



A failure/repair model

We take the modelling decision to ignore the potential failures which could occur during the very brief stages of loading and unloading jobs.

A failure/repair model

We take the modelling decision to ignore the potential failures which could occur during the very brief stages of loading and unloading jobs.

We model a failure and repair cycle taking a job back to re-execute the present stage (rather than restart the execution of the job from the beginning).

Nodes

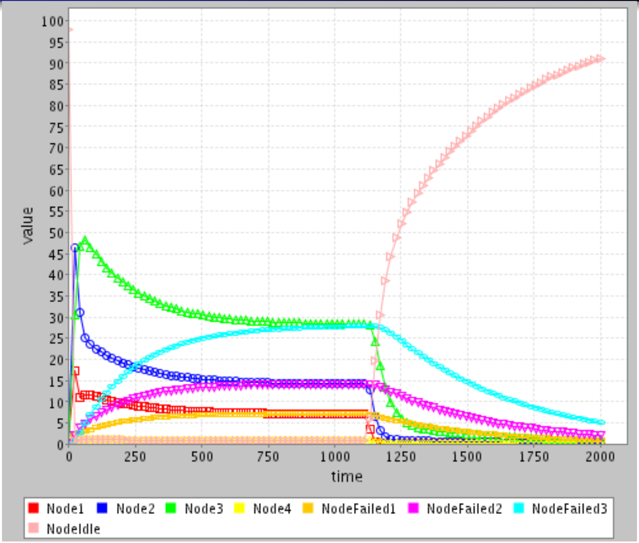
$$
\begin{aligned}
\mathit{NodeIdle} &\stackrel{\mathit{def}}{=} (\mathit{load}, r_0).\mathit{Node1} \\
\mathit{Node1} &\stackrel{\mathit{def}}{=} (\mathit{stage1}, r_1).\mathit{Node2} + (\mathit{fail1}, r_4).\mathit{NodeFailed1} \\
\mathit{Node2} &\stackrel{\mathit{def}}{=} (\mathit{stage2}, r_2).\mathit{Node3} + (\mathit{fail2}, r_4).\mathit{NodeFailed2} \\
\mathit{Node3} &\stackrel{\mathit{def}}{=} (\mathit{stage3}, r_3).\mathit{Node4} + (\mathit{fail3}, r_4).\mathit{NodeFailed3} \\
\mathit{Node4} &\stackrel{\mathit{def}}{=} (\mathit{unload}, r_0).\mathit{NodeIdle} \\
\mathit{NodeFailed1} &\stackrel{\mathit{def}}{=} (\mathit{repair1}, r_5).\mathit{Node1} \\
\mathit{NodeFailed2} &\stackrel{\mathit{def}}{=} (\mathit{repair2}, r_5).\mathit{Node2} \\
\mathit{NodeFailed3} &\stackrel{\mathit{def}}{=} (\mathit{repair3}, r_5).\mathit{Node3}
\end{aligned}
$$

Failure rates

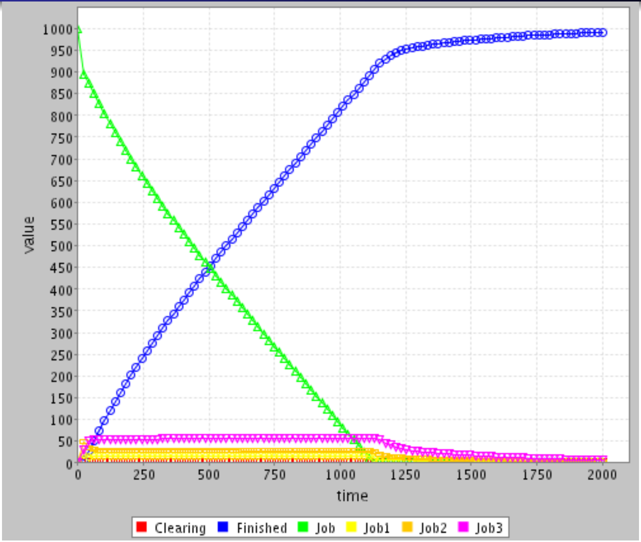
With regard to the rates of failure of jobs, we estimate that one in ten jobs may fail during stage 3 (and so one in 20 during stage 2 and one in 40 during stage 1) and that the cost of repairs is relatively high, perhaps requiring a reboot of the failed node.

Rate	Value	Interpretation
r_4	0.0025	On average 1 in 10 stage 3 jobs will fail
r_5	0.0025	Repairing may require the reboot of a node

Analysis of the failure/repair model: Nodes



Analysis of the failure/repair model: Jobs



Outline

1. Quantitative modelling with CTMCs and ODEs
   - Modelling with quantified process algebras
   - Analysis based on Continuous-time Markov Chains
   - Analysis based on Ordinary Differential Equations
2. Performance modelling with process algebras
   - Performance Evaluation Process Algebra
   - PEPA model of jobs and servers
   - Analysis of the model
3. Comparing performance measures
   - Computed with continuous time
   - Computed with continuous space
   - Comparison of computed measures
4. Commentary and comparison

Computing performance measures: CTMCs

Queue example

$$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_i \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_{i+1} + (\mathit{serve}, \mu).Q_{i-1}$$

$$Q_8 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_7 \qquad (0 < i < 8)$$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$).

Computing performance measures: CTMCs

Queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_i \stackrel{def}{=} (arrive, \lambda).Q_{i+1} + (serve, \mu).Q_{i-1}$

$Q_8 \stackrel{def}{=} (serve, \mu).Q_7 \qquad (0 < i < 8)$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$). For $\lambda = 1, \mu = 4$ steady-state is:

0. 0.7500
1. 0.1875
2. 0.0468
3. 0.0117
4. 0.0029
5. 0.0007
6. 0.0000
7. 0.0000
8. 0.0000

Computing performance measures: CTMCs

Queue example

$$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_i \stackrel{def}{=} (arrive, \lambda).Q_{i+1} + (serve, \mu).Q_{i-1}$$
$$Q_8 \stackrel{def}{=} (serve, \mu).Q_7 \qquad (0 < i < 8)$$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$). For $\lambda = 1, \mu = 2$ steady-state is:

0. 0.5009
1. 0.2504
2. 0.1252
3. 0.0626
4. 0.0313
5. 0.0156
6. 0.0078
7. 0.0039
8. 0.0019

Computing performance measures: CTMCs

Queue example

$$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_i \stackrel{def}{=} (arrive, \lambda).Q_{i+1} + (serve, \mu).Q_{i-1}$$
$$Q_8 \stackrel{def}{=} (serve, \mu).Q_7 \qquad (0 < i < 8)$$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$). For $\lambda = 1, \mu = 1$ steady-state is:

0. 0.1111
1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1111
6. 0.1111
7. 0.1111
8. 0.1111

Computing performance measures: CTMCs

Queue example

$$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_i \stackrel{def}{=} (arrive, \lambda).Q_{i+1} + (serve, \mu).Q_{i-1}$$
$$Q_8 \stackrel{def}{=} (serve, \mu).Q_7 \qquad (0 < i < 8)$$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$). For $\lambda = 2, \mu = 1$ steady-state is:

0. 0.0019
1. 0.0039
2. 0.0078
3. 0.0156
4. 0.0313
5. 0.0626
6. 0.1252
7. 0.2504
8. 0.5009

Computing performance measures: CTMCs

Queue example

$$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_i \stackrel{def}{=} (arrive, \lambda).Q_{i+1} + (serve, \mu).Q_{i-1}$$

$$Q_8 \stackrel{def}{=} (serve, \mu).Q_7 \qquad (0 < i < 8)$$

A queue with arrivals at rate λ, service at rate μ and capacity 8 (thus $0 \leq \text{len} < 9$). For $\lambda = 4, \mu = 1$ steady-state is:

0. 0.0000
1. 0.0000
2. 0.0000
3. 0.0007
4. 0.0029
5. 0.0117
6. 0.0468
7. 0.1875
8. 0.7500

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333
1	2	0.9824

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333
1	2	0.9824
1	1	4.0000

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333
1	2	0.9824
1	1	4.0000
2	1	7.0176

Calculating average queue length: CTMCs

To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333
1	2	0.9824
1	1	4.0000
2	1	7.0176
4	1	7.6667

Calculating average queue length: CTMCs

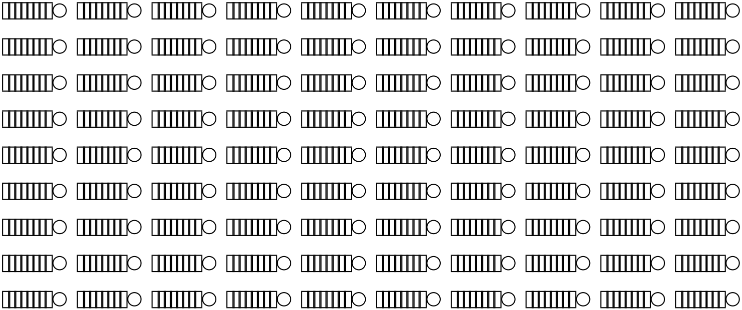
To calculate the average queue length, weight the probability of a state by the number of customers in the queue at that point.

$$a = \sum_{i=0}^{8} i\pi(i)$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at equilibrium)
1	4	0.3333
1	2	0.9824
1	1	4.0000
2	1	7.0176
4	1	7.6667

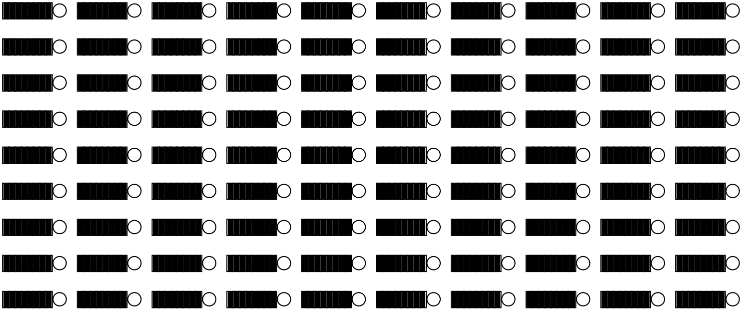
Queues and differential equations

CTMC: 

ODEs: 

Queues and differential equations

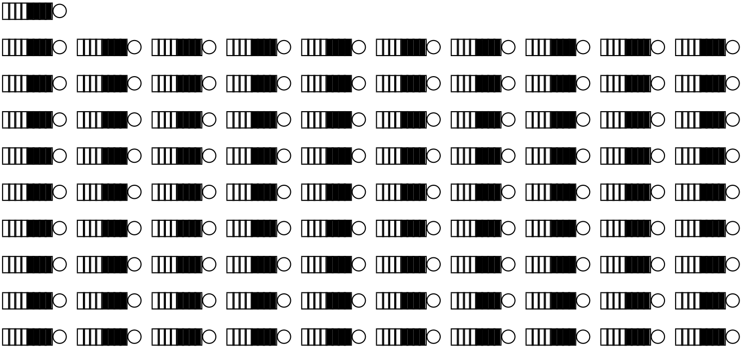
CTMC: 

ODEs: 

Queues and differential equations

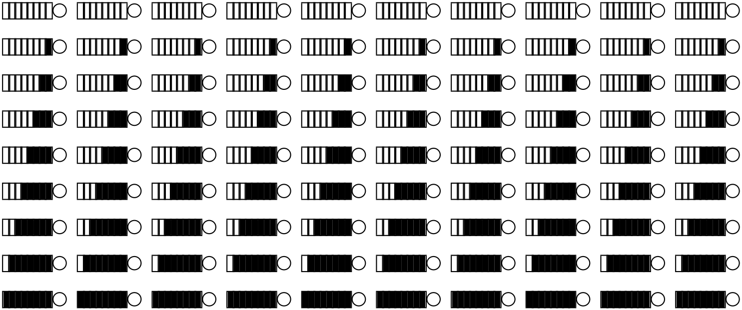
CTMC:

ODEs:

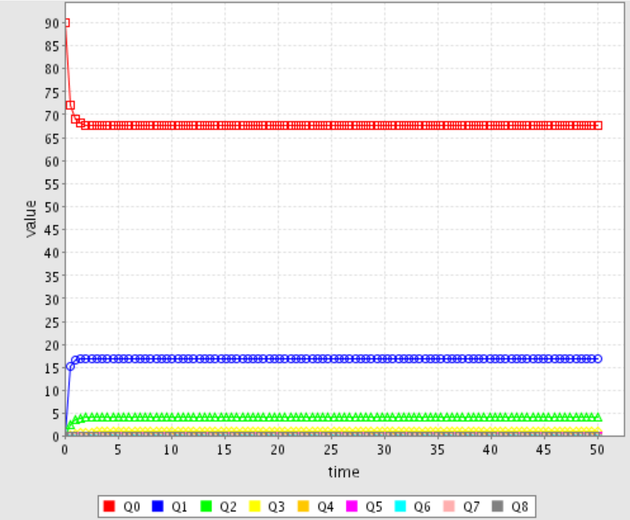


Queues and differential equations

CTMC: 

ODEs: 

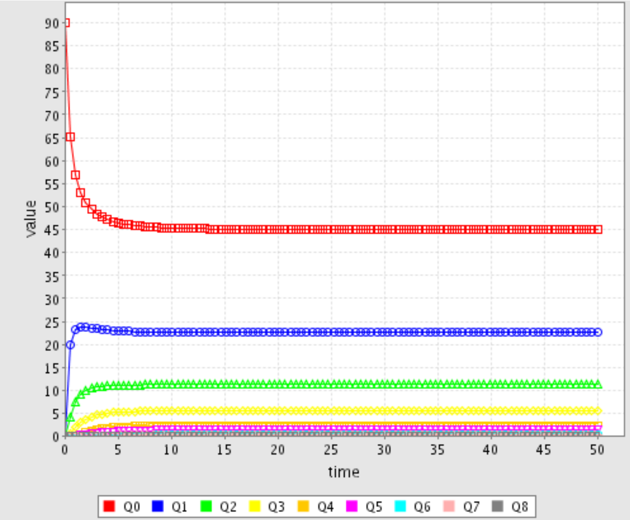
Computing performance measures: ODEs



$\lambda = 1$

$\mu = 4$

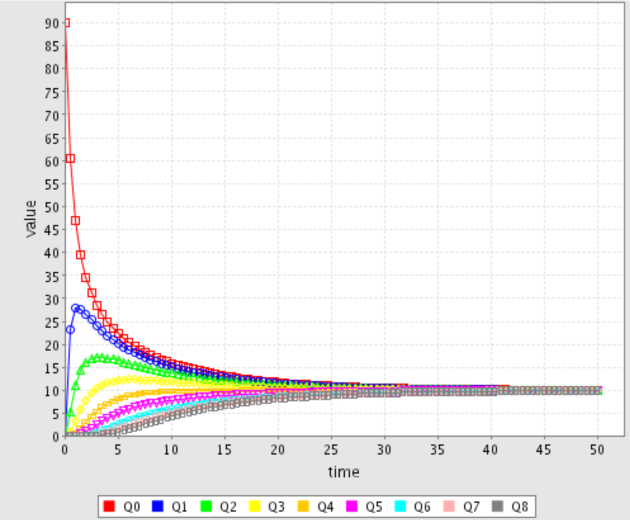
Computing performance measures: ODEs



$\lambda = 1$

$\mu = 2$

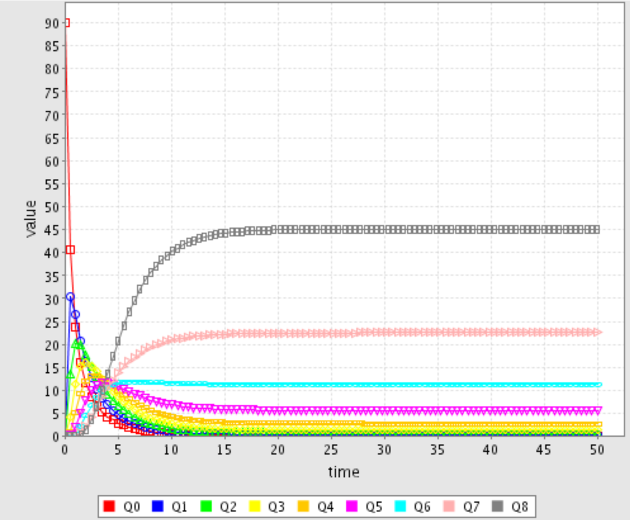
Computing performance measures: ODEs



$\lambda = 1$

$\mu = 1$

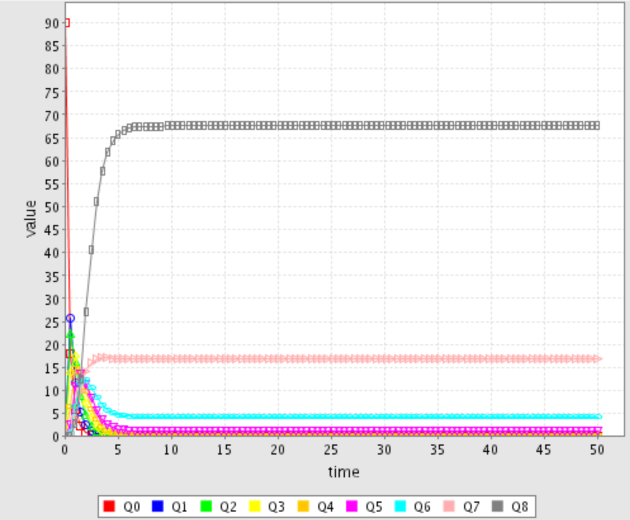
Computing performance measures: ODEs



$\lambda = 2$

$\mu = 1$

Computing performance measures: ODEs



$\lambda = 4$

$\mu = 1$

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333
1	2	0.9824

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333
1	2	0.9824
1	1	3.9914

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333
1	2	0.9824
1	1	3.9914
2	1	7.0176

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333
1	2	0.9824
1	1	3.9914
2	1	7.0176
4	1	7.6667

Calculating average queue length: ODEs

To calculate the average queue length, weight the fraction of queues of a given length by the number of customers in the queue.

$$a = \sum_{i=0}^{8} i \frac{[Q_i]}{90}$$

Arrival rate (λ)	Service rate (μ)	Av. queue length (at $t = 50$)
1	4	0.3333
1	2	0.9824
1	1	3.9914
2	1	7.0176
4	1	7.6667

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 50$)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t =$ 50)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}
1	2	0.982387959648	0.982387242222	7.1×10^{-7}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 50$)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}
1	2	0.982387959648	0.982387242222	7.1×10^{-7}
1	1	4.000000000000	3.991409877780	8.6×10^{-3}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 50$)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}
1	2	0.982387959648	0.982387242222	7.1×10^{-7}
1	1	4.000000000000	3.991409877780	8.6×10^{-3}
2	1	7.017612040350	7.017612412220	-3.7×10^{-7}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 50$)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}
1	2	0.982387959648	0.982387242222	7.1×10^{-7}
1	1	4.000000000000	3.991409877780	8.6×10^{-3}
2	1	7.017612040350	7.017612412220	-3.7×10^{-7}
4	1	7.666700990970	7.666701341490	-3.5×10^{-7}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 50$)	Difference
1	4	0.333299009029	0.333298624889	3.8×10^{-7}
1	2	0.982387959648	0.982387242222	7.1×10^{-7}
1	1	4.000000000000	3.991409877780	8.6×10^{-3}
2	1	7.017612040350	7.017612412220	-3.7×10^{-7}
4	1	7.666700990970	7.666701341490	-3.5×10^{-7}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 100$)	Difference
1	4	0.333299009029	0.333298736822	2.7×10^{-7}
1	2	0.982387959648	0.982387201111	7.6×10^{-7}
1	1	4.000000000000	3.999979511110	2.0×10^{-5}
2	1	7.017612040350	7.017613132220	-1.1×10^{-6}
4	1	7.666700990970	7.666701089580	-9.8×10^{-8}

Comparison of computed measures

λ	μ	Av. queue length (CTMCs at equilibrium)	Av. queue length (ODEs at $t = 200$)	Difference
1	4	0.333299009029	0.333298753978	2.5×10^{-7}
1	2	0.982387959648	0.982386995556	9.6×10^{-7}
1	1	4.000000000000	4.000000266670	-2.6×10^{-7}
2	1	7.017612040350	7.017613704440	-1.6×10^{-6}
4	1	7.666700990970	7.666701306580	-3.2×10^{-7}

Small queue example: CTMCs

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$

$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

Small queue example: CTMCs

Small queue example

$$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$$
$$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$$

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda & 0 \\ \mu & -\lambda-\mu & \lambda \\ 0 & \mu & -\mu \end{bmatrix}$$

Small queue example: CTMCs

Small queue example

$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$

$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda & 0 \\ \mu & -\lambda - \mu & \lambda \\ 0 & \mu & -\mu \end{bmatrix} \qquad \boxed{\pi \mathbf{Q} = \mathbf{0}}$$

Small queue example: CTMCs

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$
$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda & 0 \\ \mu & -\lambda - \mu & \lambda \\ 0 & \mu & -\mu \end{bmatrix} \qquad \boxed{\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}} \qquad \boxed{\sum \boldsymbol{\pi} = 1}$$

Small queue example: CTMCs

Small queue example

$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$

$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda & 0 \\ \mu & -\lambda-\mu & \lambda \\ 0 & \mu & -\mu \end{bmatrix} \quad \boxed{\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}} \quad \boxed{\sum \boldsymbol{\pi} = 1}$$

$$\boldsymbol{\pi} = \left[\frac{\mu^2}{\lambda^2 + \mu\,\lambda + \mu^2}, \frac{\mu\,\lambda}{\lambda^2 + \mu\,\lambda + \mu^2}, \frac{\lambda^2}{\lambda^2 + \mu\,\lambda + \mu^2} \right]$$

Small queue example: ODEs

Small queue example

$$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$$
$$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$$

Small queue example: ODEs

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$

$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

$$\frac{dQ_0}{dt} = -\lambda Q_0 + \mu Q_1$$

Small queue example: ODEs

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$
$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

$$\begin{aligned}
\frac{dQ_0}{dt} &= -\lambda Q_0 + \mu Q_1 \\
\frac{dQ_1}{dt} &= \lambda Q_0 - \lambda Q_1 - \mu Q_1 + \mu Q_2
\end{aligned}$$

Small queue example: ODEs

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$

$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

$$\begin{aligned}
\frac{dQ_0}{dt} &= -\lambda Q_0 + \mu Q_1 \\
\frac{dQ_1}{dt} &= \lambda Q_0 - \lambda Q_1 - \mu Q_1 + \mu Q_2 \\
\frac{dQ_2}{dt} &= \lambda Q_1 - \mu Q_2
\end{aligned}$$

Small queue example: ODEs (stationary points)

Small queue example

$Q_0 \stackrel{def}{=} (arrive, \lambda).Q_1 \qquad Q_1 \stackrel{def}{=} (arrive, \lambda).Q_2 + (serve, \mu).Q_0$

$Q_2 \stackrel{def}{=} (serve, \mu).Q_1$

$$
\begin{aligned}
0 &= -\lambda Q_0 + \mu Q_1 \\
0 &= \lambda Q_0 - \lambda Q_1 - \mu Q_1 + \mu Q_2 \\
0 &= \lambda Q_1 - \mu Q_2
\end{aligned}
$$

Small queue example: ODEs (stationary points)

Small queue example

$$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$$

$$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$$

$$\mathbf{0} = [Q_0 \quad Q_1 \quad Q_2] \begin{bmatrix} -\lambda & \lambda & 0 \\ \mu & -\lambda - \mu & \lambda \\ 0 & \mu & -\mu \end{bmatrix}$$

Small queue example: ODEs (and CTMC solution)

Small queue example

$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$

$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$

$$\mathbf{p} \quad = \quad [Q_0 \quad \frac{\lambda}{\mu} Q_0 \quad \frac{\lambda^2}{\mu^2} Q_0]$$

Small queue example: ODEs (and CTMC solution)

Small queue example

$$Q_0 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_1 \qquad Q_1 \stackrel{\mathit{def}}{=} (\mathit{arrive}, \lambda).Q_2 + (\mathit{serve}, \mu).Q_0$$

$$Q_2 \stackrel{\mathit{def}}{=} (\mathit{serve}, \mu).Q_1$$

$$\begin{aligned}
\mathbf{p} &= [Q_0 \quad \frac{\lambda}{\mu}Q_0 \quad \frac{\lambda^2}{\mu^2}Q_0] \\
\boldsymbol{\pi} &= \left[\frac{\mu^2}{\lambda^2 + \mu\,\lambda + \mu^2}, \frac{\mu\,\lambda}{\lambda^2 + \mu\,\lambda + \mu^2}, \frac{\lambda^2}{\lambda^2 + \mu\,\lambda + \mu^2}\right]
\end{aligned}$$

What just happened?

We found that, for a sequential PEPA component, the differential equations are recording the same information as found in the infinitesimal generator matrix of the Markov chain.

What just happened?

We found that, for a sequential PEPA component, the differential equations are recording the same information as found in the infinitesimal generator matrix of the Markov chain.

The stationary points of the system of ODEs for an initial value of 1 make up the stationary probability distribution of the CTMC.

Isn't this just the Chapman-Kolmogorov equations?

Now that we have discovered that we have a copy of a generator matrix in the ODEs aren't we just back to

$$\frac{d\pi(t)}{dt} = \pi(t)Q \ ?$$

Isn't this just the Chapman-Kolmogorov equations?

Now that we have discovered that we have a copy of a generator matrix in the ODEs aren't we just back to

$$\frac{d\pi(t)}{dt} = \pi(t)Q \ ?$$

Only if the system is a single sequential component. For even only two parallel queues, the generator matrix is much larger than the system of ODEs.

Generator matrix for two parallel queues

$$\mathbf{Q} = \begin{bmatrix} -2\lambda & \lambda & \lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ \mu & -2\lambda-\mu & 0 & \lambda & \lambda & 0 & 0 & 0 & 0 \\ \mu & 0 & -2\lambda-\mu & 0 & \lambda & 0 & 0 & 0 & \lambda \\ 0 & \mu & 0 & -\lambda-\mu & 0 & \lambda & 0 & 0 & 0 \\ 0 & \mu & \mu & 0 & -2\lambda-2\mu & \lambda & 0 & \lambda & 0 \\ 0 & 0 & 0 & \mu & \mu & -\lambda-2\mu & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu & -2\mu & \mu & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 & \lambda & -\lambda-2\mu & \mu \\ 0 & 0 & \mu & 0 & 0 & 0 & 0 & \lambda & -\lambda-\mu \end{bmatrix}$$

Steady-state for two parallel queues

$$\pi = \begin{bmatrix} \frac{\mu^4}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu^3\lambda}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu^3\lambda}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu^2\lambda^2}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu^2\lambda^2}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu\lambda^3}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\lambda^4}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu\lambda^3}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4}, \\ \frac{\mu^2\lambda^2}{2\mu\lambda^3+3\mu^2\lambda^2+2\mu^3\lambda+\lambda^4+\mu^4} \end{bmatrix}$$

Outline

1. Quantitative modelling with CTMCs and ODEs
   - Modelling with quantified process algebras
   - Analysis based on Continuous-time Markov Chains
   - Analysis based on Ordinary Differential Equations
2. Performance modelling with process algebras
   - Performance Evaluation Process Algebra
   - PEPA model of jobs and servers
   - Analysis of the model
3. Comparing performance measures
   - Computed with continuous time
   - Computed with continuous space
   - Comparison of computed measures
4. Commentary and comparison

Commentary and comparison

- Previous performance modelling with PEPA used continuous-time Markov chains (CTMCs). These admit *steady-state* and *transient* analysis (by solving the CTMC).

Commentary and comparison

- Previous performance modelling with PEPA used continuous-time Markov chains (CTMCs). These admit *steady-state* and *transient* analysis (by solving the CTMC).
- Steady-state is cheaper but less informative. Transient is more informative but more expensive.

Commentary and comparison

- Previous performance modelling with PEPA used continuous-time Markov chains (CTMCs). These admit *steady-state* and *transient* analysis (by solving the CTMC).
- Steady-state is cheaper but less informative. Transient is more informative but more expensive.
- Major drawback: state-space explosion. Generating the state-space is slow. Solving the CTMC is slow.

Commentary and comparison

- Previous performance modelling with PEPA used continuous-time Markov chains (CTMCs). These admit *steady-state* and *transient* analysis (by solving the CTMC).
- Steady-state is cheaper but less informative. Transient is more informative but more expensive.
- Major drawback: state-space explosion. Generating the state-space is slow. Solving the CTMC is slow.
- In practice effective only to systems of size 10^6 states, even when using clever storage representations.

Commentary and comparison

- Mapping PEPA to ODEs admits *course-of-values* analysis by solving the ODE (akin to transient analysis).

Commentary and comparison

- Mapping PEPA to ODEs admits *course-of-values* analysis by solving the ODE (akin to transient analysis).
- Major benefit: avoids state-space generation entirely.

Commentary and comparison

- Mapping PEPA to ODEs admits *course-of-values* analysis by solving the ODE (akin to transient analysis).
- Major benefit: avoids state-space generation entirely.
- Major benefit: ODE solving is effective in practice, leaning towards suitability for interactive experimentation. Good for modellers, gaining more insights into the system behaviour.

Commentary and comparison

- Mapping PEPA to ODEs admits *course-of-values* analysis by solving the ODE (akin to transient analysis).
- Major benefit: avoids state-space generation entirely.
- Major benefit: ODE solving is effective in practice, leaning towards suitability for interactive experimentation. Good for modellers, gaining more insights into the system behaviour.
- Effective for systems of size 10^{10^6} states and beyond.

Discussion: process algebras and ODEs

- Models in the PEPA stochastic process algebra are concise, and in direct style they generate a system of ODEs the number of which is linear in the number of distinct component types in the PEPA model.

Discussion: process algebras and ODEs

- Models in the PEPA stochastic process algebra are concise, and in direct style they generate a system of ODEs the number of which is linear in the number of distinct component types in the PEPA model.
- Thus there is no hidden cost in the use of the high-level language but there are many advantages.

Discussion: process algebras and ODEs

- Models in the PEPA stochastic process algebra are concise, and in direct style they generate a system of ODEs the number of which is linear in the number of distinct component types in the PEPA model.
- Thus there is no hidden cost in the use of the high-level language but there are many advantages.
    - PEPA models can be checked for freedom from deadlock, satisfaction of logical properties, or compared using relations such as bisimulation.

Discussion: process algebras and ODEs

- Models in the PEPA stochastic process algebra are concise, and in direct style they generate a system of ODEs the number of which is linear in the number of distinct component types in the PEPA model.
- Thus there is no hidden cost in the use of the high-level language but there are many advantages.
  - PEPA models can be checked for freedom from deadlock, satisfaction of logical properties, or compared using relations such as bisimulation.
  - As a compositional modelling language PEPA components can be re-used in other models, promoting best practice.

Commentary and comparison

- Analysis capabilities
  - Numerical integration, course-of-values analysis
  - Interested in the solution of *initial value problems*
  - Interested in finding *stationary points*
  - Verification at process algebra level (freedom from deadlock)

Commentary and comparison

- Analysis capabilities
  - Numerical integration, course-of-values analysis
  - Interested in the solution of *initial value problems*
  - Interested in finding *stationary points*
  - Verification at process algebra level (freedom from deadlock)
- Relationship to other analysis methods
  - For sequential components, we can understand the relationship between the CTMC and ODE solution via the (same) generator matrix.
  - No such relationship exists for stochastic simulation and Markov chains.

False and true concurrency

In the process algebra world, algebras with an interleaving semantics are termed false concurrency. PEPA [Hillston 1994] was the first timed process algebra to have an interleaving semantics allowing it to generate a CTMC. The interleaving semantics gives rise to the state-space explosion problem.

False and true concurrency

In the process algebra world, algebras with an interleaving semantics are termed false concurrency. PEPA [Hillston 1994] was the first timed process algebra to have an interleaving semantics allowing it to generate a CTMC. The interleaving semantics gives rise to the state-space explosion problem.

Process algebras without an interleaving semantics are termed true concurrency process algebras. The search for a true concurrency timed process algebra has been a ten-year open problem.

False and true concurrency

In the process algebra world, algebras with an interleaving semantics are termed false concurrency. PEPA [Hillston 1994] was the first timed process algebra to have an interleaving semantics allowing it to generate a CTMC. The interleaving semantics gives rise to the state-space explosion problem.

Process algebras without an interleaving semantics are termed true concurrency process algebras. The search for a true concurrency timed process algebra has been a ten-year open problem.

PEPA [Hillston 2005] is the first timed process algebra to have a true concurrency semantics via the mapping to ODEs. The true concurrency semantics avoids the state-space explosion problem.