Enhancing the effective utilisation of Grid clusters by exploiting on-line performability analysis

Stephen Gilmore
LFCS, Edinburgh

Joint work with Anne Benoit, Murray Cole and Jane Hillston

Outline

Outline

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
- scheduling and re-scheduling

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
- scheduling and re-scheduling
- scale

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
- scheduling and re-scheduling
- scale
- failures

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
  - provide a modelling language to describe diverse systems
- scheduling and re-scheduling
- scale
- failures

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
  - provide a modelling language to describe diverse systems
- scheduling and re-scheduling
  - provide a low-cost analysis of time-dependent behaviour
- scale

- failures

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
  - provide a modelling language to describe diverse systems
- scheduling and re-scheduling
  - provide a low-cost analysis of time-dependent behaviour
- scale
  - provide a parametric analysis which scales to large job sizes
- failures

Challenges of Grid performability

Grid computing is characterised by

- heterogeneity
    - provide a modelling language to describe diverse systems
- scheduling and re-scheduling
    - provide a low-cost analysis of time-dependent behaviour
- scale
    - provide a parametric analysis which scales to large job sizes
- failures
    - allow custom recovery procedures to be specified

Addressing the challenges

- Use a high-level programming model to structure code.
- Use a high-level modelling language to analyse performance.

Addressing the challenges

- Use a high-level programming model to structure code.
  - *Cole's "algorithmic skeletons" (eSkel library)*
- Use a high-level modelling language to analyse performance.

Addressing the challenges

- Use a high-level programming model to structure code.
  - *Cole's "algorithmic skeletons" (eSkel library)*
- Use a high-level modelling language to analyse performance.
  - *Hillston's Performance Evaluation Process Algebra (PEPA)*

Modelling and analysis

- We describe the workload and the computing fabric as a high-level model in PEPA.

Modelling and analysis

- We describe the workload and the computing fabric as a high-level model in PEPA.
- We map PEPA models into ordinary differential equations (ODEs) for solution.

Modelling and analysis

- We describe the workload and the computing fabric as a high-level model in PEPA.
- We map PEPA models into ordinary differential equations (ODEs) for solution.
- The analysis is supported by an automated tool which handles the transformation from the high-level process algebra model into ODEs and numerical integration.

Modelling and analysis

- We describe the workload and the computing fabric as a high-level model in PEPA.
- We map PEPA models into ordinary differential equations (ODEs) for solution.
- The analysis is supported by an automated tool which handles the transformation from the high-level process algebra model into ODEs and numerical integration.
- The results are returned to the user as a plot of the numbers of model components as a function of time.

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

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

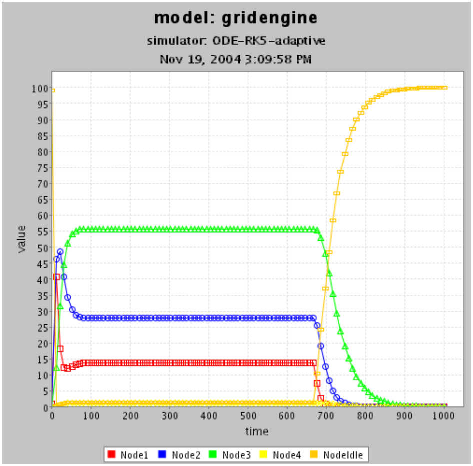
where L is $\{\, \mathit{load}, \mathit{stage1}, \mathit{stage2}, \mathit{stage3}, \mathit{unload} \,\}$.

Analysis of the model

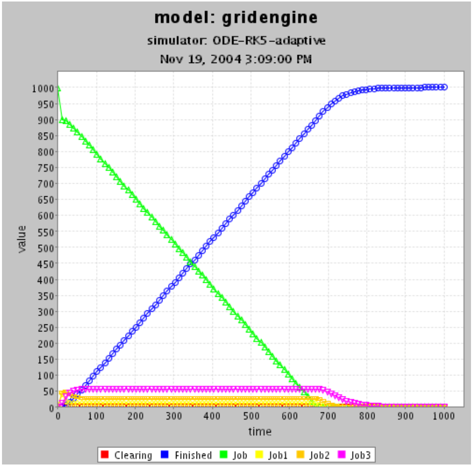
Analysis of the model proceeds by choosing particular values for the rates. The values below are chosen to make the analysis easy to follow.

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



Outline

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

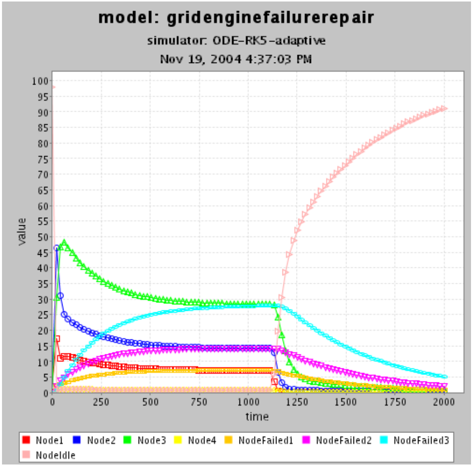
$$
\begin{aligned}
\mathit{NodeIdle} &\stackrel{\mathit{def}}{=} (\mathit{load}, r_0).\mathit{Node1} \\
\mathit{Node1} &\stackrel{\mathit{def}}{=} (\mathit{stage1}, r_1).\mathit{Node2} + (\mathit{fail1}, r_4).\mathit{NodeFailed1} \\
\mathit{Node2} &\stackrel{\mathit{def}}{=} (\mathit{stage2}, r_2).\mathit{Node3}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

Failure rates

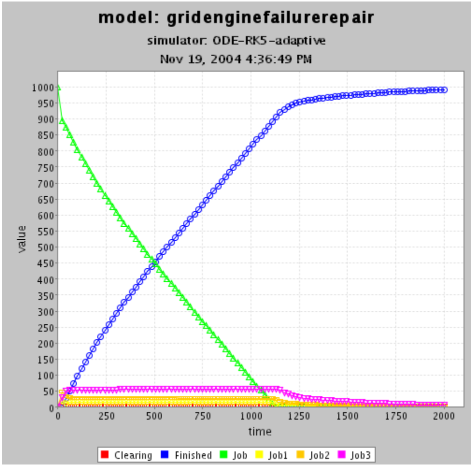
With regard to the rates of failure of jobs, we estimate that one in ten jobs may fail during stage 3 (and so one in 20 during stage 2 and one in 40 during stage 1) and that the cost of repairs is relatively high, perhaps requiring a reboot of the failed node.

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

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. (Suitable for on-line scheduling?)

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. (Suitable for on-line scheduling?)
- Effective for systems of size 2^{10^6} states and beyond.