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

Anne Benoit, Murray Cole, Stephen Gilmore and Jane Hillston
School of Informatics
The University of Edinburgh
Edinburgh EH9 3JZ
Email: enhancers@inf.ed.ac.uk

Abstract

In Grid applications the heterogeneity and potential failures of the computing infrastructure poses significant challenges to efficient scheduling. Performance models have been shown to be useful in providing predictions on which schedules can be based [1, 2] and most such techniques can also take account of failures and degraded service. However, when several alternative schedules are to be compared it is vital that the analysis of the models does not become so costly as to outweigh the potential gain of choosing the best schedule. Moreover, it is vital that the modelling approach can scale to match the size and complexity of realistic applications.

In this paper we present a novel method of modelling job execution on Grid compute clusters. As previously we use Performance Evaluation Process Algebra (PEPA) [3] as the system description formalism, capturing both workload and computing fabric. The novel feature is that we make a continuous approximation of the state space underlying the PEPA model and represent it as a set of ordinary differential equations (ODEs) for solution, rather than a continuous time, but discrete state space, Markov chain.

1 Introduction

Grid engines such as Condor [4] and Sun Grid Engine (SGE) [5] allow users to queue up jobs for execution on cluster or grid technology to be executed when their necessary data and hardware resources become available. The best grid engines provide rich job control languages and a sophisticated queueing agent. Grid engines are deployed on compute clusters such as Beowulfs in order to share computational resources across a number of, possibly interdependent, jobs [6].

Analysing high-level models of program execution on Grid clusters allows underused resources to be more efficiently deployed. This benefits users working with computationally-intensive problems such as those found in the physical sciences. For such an analysis to be effective, the analysis process itself should have low computational cost, otherwise it could impede the execution of the genuine computational load. For such an analysis to be widely applicable it should scale to be able to model compute clusters with a sizeable number of processes, each of which is executing a large number of compound jobs, made up of numerous stages. For the analysis to be useful the models need to faithfully represent the inevitable software and hardware failures which will occur at some time while executing in a sequence of long-running computations. The intention of the analysis should be to model realistic Grid configurations, not idealised or simplified versions of these, and to do this analysis inexpensively.

Many analysis methods in current use do not address all of the above requirements well. Some methods are scalable, but have long running times: simulation-based methods [7, 8, 9] and genetic algorithm-based methods [10] could be considered to be in this category. As an analysis method simulation has the disadvantage that it leaves the additional burden of needing to compute con-

fidence intervals for the results. Other analysis methods have efficient solution procedures but do not scale well: Continuous-Time Markov Chains (CTMCs) as used in [2] are in this category. Still other methods make strong simplifying assumptions or introduce gross approximations which compromise their accuracy.

One approach to the problem of scalability is to use a continuous approximation of the discrete state space underlying the mathematical representation of the model. For example, if there are 1000 jobs to be processed it may be useful to think of a continuous flow of jobs, rather than singly representing each of the individual states as measures such as *the number of completed jobs* grow in discrete unit steps. This is an approximation since it may lead to the model being in a state in which 55.8 jobs have completed which has no analogue in real life. However, such systems can be readily represented as a set of ordinary differential equations (ODEs). Solving such a system has low computational cost and modest memory requirements when the system is concise.

This is a significant advantage but suggesting to use ODEs directly as a modelling language for this application would be a questionable one. ODEs would be unfamiliar to most of the practicising system managers who are charged with running Grid services. For this reason, rather than work directly with ODEs we model with a high-level language of recursively defined communicating finite-state processes (the PEPA process algebra [3]), and generate ODEs from this language [11].

Models in the PEPA stochastic process algebra are concise, and under the application of Hillston's method [11], 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. Using other software tools [12, 13, 14], PEPA models can be checked for freedom from deadlock, satisfaction of logical properties, or solved for steady-state or transient measures. Verification procedures such as these are available for process algebras but not for ODEs, so the use of a high-level language confers additional benefits above working with ODEs directly.

ODEs can be solved numerically using solvers which implement the Runge-Kutta method, or Rosenbrock's algorithm, or others. Numerical computing platforms offer high-level support for the solution of ODEs [15]. We are interested in the solution of *initial value problems* (IVPs) where the initial quantities of the components of the problem are known and we wish to find out how these change over time. Compared with modelling with CTMCs, modelling with ODEs resembles most strongly transient analysis of CTMCs: there is no implicit assumption that the system reaches steady-state equilibrium and we observe states of the system as time progresses, working forwards from their initial values at time $t = 0$.

As noted by Gillespie and others [16, 17, 18, 19], the differential equation approach is applicable when there are sufficiently large numbers of each interacting entity in the model (in our case these entities are jobs and servers). This makes this modelling approach particularly applicable to Grid-scale computing with large numbers of jobs executing on large compute clusters. In cases of only a small number of jobs executing on a small number of processors other analysis methods may be more accurate, including stochastic simulation [16, 18] or CTMC-based solution. These methods are already available for the PEPA stochastic process algebra in tools such as the PEPA Workbench [12], Möbius [20], PRISM [13] and The Imperial PEPA Compiler (IPC) [14]. Our differential equation-based analysis complements these, and allows PEPA modelling to be applied to systems which are significantly larger than those which can be modelled by simulation or CTMCs.

The original contribution of the present paper is that it is the first to report on the benefits of mapping stochastic process algebras to ordinary differential equations for analysis instead of to continuous-time Markov chains, semi-Markov processes or generalised semi-Markov processes. In addition, we believe it to be the first paper to show the potential for ODEs, however they are obtained, to be used as a modelling tool for Grid compute clusters. We suggest that this is particularly valuable for making rapid performance predictions to be used when on-line scheduling and re-scheduling decisions have to be made.

Structure of this paper: Section 2 presents an introduction to Performance Evaluation Process Algebra. Section 3 presents a simple model of jobs and servers and discusses its mapping to a system of ODEs. Section 4 presents the model extended with failures and repairs. A discussion of related work and conclusions follow.

2 PEPA

We present a brief introduction to PEPA to make the present paper self-contained. For full details the reader is referred to [3].

In PEPA modelling a system is viewed as a set of *components* which carry out *activities* either individually or in cooperation with other components. Activities which are private to the component in which they occur are represented by the distinguished action type, τ. Each activity is characterized by an *action type* and a rate. This is written as a pair such as (α, r) where α is the action type and r is the *activity rate*. This parameter may be any positive real number, or may be unspecified. We use the distinguished symbol $\top$ to indicate that the rate is not specified by this component. This component is said to be *passive* with respect to this action type and the rate of the shared activity is defined by another component.

PEPA provides a set of combinators which allow expressions to be built which define the behaviour of components via the activities that they engage in. These combinators are presented below.

Prefix: $(\alpha, r).P$: Prefix is the basic mechanism by which the behaviours of components are constructed. This combinator implies that after the component has carried out activity (α, r), it behaves as component P.

Choice: $P_1 + P_2$: This combinator represents a competition between components. The system may behave either as component P_1 or as P_2. All current activities of the two components are enabled. The first activity to complete distinguishes one of these components and the other is then discarded.

Cooperation: $P_1 \underset{L}{\bowtie} P_2$: This describes the synchronization of components P_1 and P_2 over the activities in the cooperation set L. The components may proceed independently with activities whose types do not belong to this set. A particular case of the cooperation is when $L = \emptyset$. In this case, components proceed with all activities independently. The notation $P_1 \parallel P_2$ is used as a shorthand for $P_1 \underset{\emptyset}{\bowtie} P_2$. In a cooperation, the rate of a shared activity is defined as the rate of the slowest component.

Hiding: P/L This component behaves like P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \stackrel{def}{=} P$ Constants are components whose meaning is given by a defining equation: $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P. This is how we assign names to components (behaviours). An explicit recursion operator is not provided but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

When PEPA is used to generate a CTMC model the activity rate r is interpreted as the parameter of an exponential distribution, the duration of an activity being a random variable. When PEPA is used to generate a system of ODEs the activity rate is interpreted as a constant rate of change.

2.1 Derived forms and additional syntax

We now describe some additional *derived forms* ("syntactic sugar") for PEPA. These do not add any expressive power to the language or require any semantic rules in addition to those in [3]. We have seen one derived form already: $P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

When we are interested in transient behaviour we use the deadlocked process *Stop* as defined in [21] to signal a component which performs no further actions. We consider this to be simply an abbreviation for a deadlocked process, as shown below.

$$\mathit{Stop} \stackrel{def}{=} \Big(\big((a, r).\mathit{Stop}\big) \underset{\{a,b\}}{\bowtie} \big((b, r).\mathit{Stop}\big) \Big) / \{ a, b \}$$

Because we will be working with large numbers of jobs and servers, we introduce another abbreviation: we write $P[n]$ to denote n copies of component P executing in parallel. For example,

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

and refer to such an abbreviation as an *array* of components.

We can add another dimension to the array. The three copies of the component P in $P[3][a, b]$ are required to synchronise on the actions a and b. Thus,

$$P[3][a, b] \equiv ((P \underset{\{a,b\}}{\bowtie} P) \underset{\{a,b\}}{\bowtie} P).$$

2.2 Interpreting a PEPA model as ODEs

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.

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. 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.

3 Modelling jobs and servers

We begin to describe modelling jobs and servers, starting with a simplistic model which we will improve by adding more realistic detail later. Consider jobs with a number of ordered stages. Here we consider jobs which consist of three ordered stages. 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. At that stage the job is finished. The PEPA model is below.

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

Each of these components will correspond to a continuous variable in the system of ODEs. Note that in this model they do not determine the rate at which the stages are completed, because they are not supplying the computational effort to achieve this. To find the specification of those values we must look at the definition of a node, which executes jobs.

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

Similarly here each component will generate a distinct ODE modelling its evolution.

In the example we consider jobs where the first stage is the cheapest, the second takes twice as long as the first and the third takes twice as long as the second. Loading and unloading have equal, unit, cost.

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

We could add more stages to jobs or vary these rates to model jobs with a different execution profile, as necessary. The stages are chosen to have the above simple relationship in their cost in order to assist the readers' intuitive interpretation of the results computed below (rather than to represent realistic Grid compute jobs).

We consider the system initiated as

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

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

Thus in the ODE system the variables corresponding to *NodeIdle* and *Job* will have the initial values shown; all other variables will be set to zero. Our tool automatically generates the corresponding system of ODEs taking the PEPA model as input and producing input suitable for a third party differential equation solution tool.

Jobs differ from nodes in the obvious way, in that the rate at which jobs are executed depends on the number of nodes but not on the number of jobs, save that this must be greater than zero. (Adding more jobs to the queue cannot

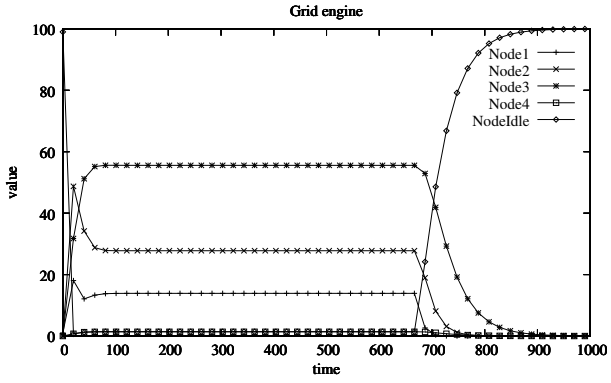

Figure 1: Plot showing the stages of utilisation of the nodes while executing 1000 jobs on a cluster of 100 nodes. All nodes are initially and finally idle.

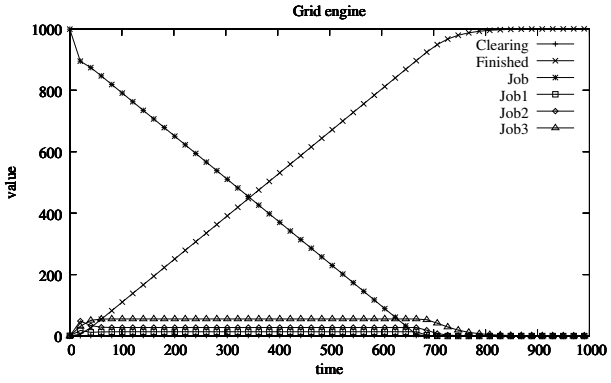

Figure 2: Plot showing the number of jobs of each stage while executing 1000 jobs on a cluster of 100 nodes. Jobs are successively completed until all 1000 are finished.

increase the collective rate at which they are completed, but adding more nodes to the cluster can.)

We compute the collective rate for executing a stage as a function of the computational cost of jobs of this class (the r_i above), the number of nodes at stage i, (N_i), and a function on the number of jobs of class i, (J_i):

$$r_i \times \min(N_i, J_i)$$

Our differential equation solution tool does not support the $\min$ function so we approximate this by

$$r_i \times N_i \times \theta(J_i)$$

where the θ function[1] is used to disable execution of jobs of class i, when there are none (by setting the rate to 0). The θ function is below:

$$\theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The analysis of the model is shown in Figures 1 and 2. Figure 1 shows the average number of nodes of each stage as a function of time. Because the second stage of each job is twice as expensive to compute as the first, there are on average twice as many nodes executing the second stage of a job at any time, and the same comparison holds for the second and third stages.

[1] Some efficient stochastic simulation algorithms, such as "tau-leap" algorithms [22] cannot be used on models containing non-differentiable functions such as θ.

Figure 2 shows that jobs progress through the system in linear fashion. We note that the number of unstarted jobs drops quickly as the nodes first become loaded (the sharp drop at the start of the graph) and then decreases steadily as jobs are successively completed. The rate at which the remaining stage 3 jobs are completed decreases as the supply of jobs runs out and more nodes are idle.

4 A failure/repair model

We now present a model which represents the failures and repairs of nodes. Such a model needs to describe at what points in the system evolution failures can occur, and identify the consequences of the failures. For example, is a job lost entirely if its host node fails, or just the current stage? Formal languages such as process algebras are well suited to this task, making explicit consequences which might be underspecified in other modelling approaches, even for widely-studied systems [23].

Here 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). The modified PEPA description of a node is below. The definition of jobs remains unchanged.

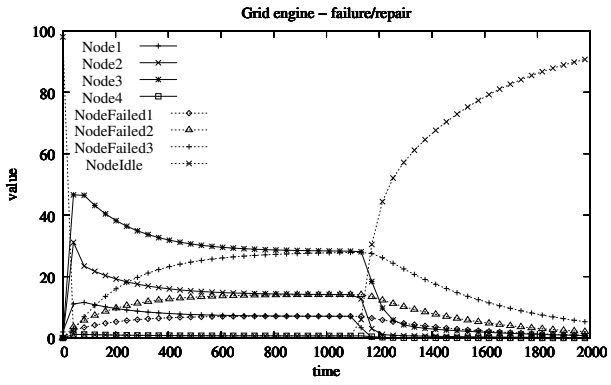

Figure 3: Plot showing the stages of utilisation of the nodes while executing 1000 jobs on a cluster of 100 nodes. Nodes which fail at each stage are recorded. All nodes are initially and finally functioning and idle.

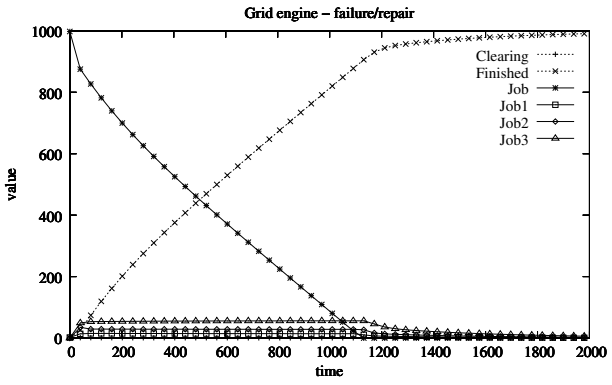

Figure 4: Plot showing the number of jobs of each stage while executing 1000 jobs on a cluster of 100 nodes in the failure/repair model. Jobs are restarted from the stage where they failed.

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

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. We model the repair process being automatically initiated after failure, without the need for operator intervention.

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

The results of evaluating this model are shown in Figures 3 and 4.

Evidently, in the presence of failures the makespan for the jobs is considerably longer. We also see the number of failed nodes rising from the initial configuration where all nodes are initially functioning. The system reaches local equilibrium just before the supply of jobs runs out. We can see that the rate at which jobs are collectively being completed drops off as more and more nodes become idle towards the end of the run.

5 Related work

Grid computing has emerged as a potential next generation platform for solving large-scale problems in many fields, involving millions of heterogeneous resources scattering across multiple locations. A sophisticated approach must be taken to analyse and fine-tune the algorithms before applying them to the real systems. Simulation appears to be the most commonly used way to analyse

algorithms on large-scale distributed systems of heterogeneous resources. Several simulators have therefore been conceived and developed for Grids. Thus, the GridSim toolkit [7] allows modelling and simulation of entities in parallel and distributed computing for the design and evaluation of scheduling algorithms. Its primary objective is to investigate effective resource allocation techniques based on computational economy through simulation. Other simulation tools aim to design and evaluate Grid middleware, applications and network services for the computational Grid, as for example MicroGrid [9]. OptorSim [8] focuses on data management, exploring the stationary and transient behaviour of several optimisation techniques to reduce data access costs.

Our objectives differ in that we are interested in quite abstract models with no concern for data. This work is being developed under the auspices of the ENHANCE project in which Grid application are structured using algorithmic skeletons[24], a PEPA template component being developed for each skeleton. The characteristics of a particular application and the current state of the Grid infrastructure are used to generate and parameterise a PEPA model which is used to investigate different possible mappings of tasks to processors.

The approach of resorting to a continuous approximation of a state space composed of many discrete entities has previously been applied to performance models in the context of both queues (e.g. [25]) and stochastic Petri nets [26]. Other authors have applied ODEs directly to work-stealing algorithms executing in a multi-processor environment [27] and distributed load-balancing using greedy algorithms [28]. To the best of our knowledge the use of continuous approximation has not been previously applied to process algebra models.

6 Conclusions

Ordinary differential equations represent formally the rate of change of populations with respect to time. Both decreasing and increasing change can be represented (for example here, as nodes fail and are repaired, respectively). ODEs introduce an approximation in that the relative probabilities of decrease and increase can be more accurately represented at larger population sizes.

Large population sizes are disadvantageous to many modelling approaches but they are the conditions under which the approximation introduced by the use of ODEs induces the least error. Large populations of computational nodes and compute jobs are characteristic of Grid computing, so the approach is applicable there.

ODEs are a deterministic modelling formalism where the duration of events is constant. This contrasts markedly with stochastic modelling with exponential distributions and we need to be cautious about whether or not this is appropriate for the modelling which we are undertaking.

Grid compute clusters are frequently used for scientific and numerical computing. This has as its building blocks numerical routines such as matrix vector multiplication. These algorithms have low data dependency and predictable running times. So much so, in fact, that modelling these by constant distributions may be more accurate than modelling them by exponentials. We have no analytical or experimental evidence to suggest that the run-time of a routine which multiplies a matrix by a vector on the right varies according to an exponential distribution.

The modelling reported here has the potential to scale to larger, more complex systems, leading towards realistic modelling of Grid computations. Our present model of Grid nodes executing multi-stage jobs is only a proof-of-concept of the usability of the analysis method, encouraging us to more forwards towards more realistic models. How best to apply the results of the modelling also remains as future work, as does fully understanding the range of applicability of the modelling technique used here.

Acknowledgements

The authors are supported by the Enhance project (“Enhancing the Performance Predictability of Grid Applications with Patterns and Process Algebras”) funded by the EPSRC under grant number GR/S21717/01. The authors thank Paul Jackson of the School of Informatics for helpful advice on modelling with ODEs. Professor Muffy Calder of the University of Glasgow provided the initial stimulus to consider mapping stochastic process algebras to ODEs.

References

[1] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. ICENI: Optimisation of Component Applications within a Grid Environment. *Parallel Computing*, 28(12):1753–1772, 2002.

[2] A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Evaluating the performance of skeleton-based high level parallel programs. In *ICCS 2004: Proc. of 4th Intl. Conference on Computational Science, Kraków, Poland*, volume 3038 of *LNCS*, pages 289–296. Springer-Verlag, 2004.

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

[4] Condor project home page. `http://www.cs.wisc.edu/condor/`, November 2004.

[5] SUN Microsystems. SUN Grid Engine Web site. `http://gridengine.sunsource.net/`, November 2004.

[6] E. L. Haletky and P. Lampert. Beowulf batch processors and job schedulers. *Systems Administration—Clustering supplement*, pages 13–17, August 2003.

[7] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), 2002.

[8] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4), 2003.

[9] Song, Liu, Jakobsen, Bhagwan, Zhang, Taura, and Chien. The MicroGrid: a Scientific Tool for Modeling Computational Grids. *Scientific Programming*, 8(3):127–141, 2000.

[10] D.P. Spooner, J. Cao, S.A. Jarvis, L. He, and G.R. Nudd. Performance-aware Workflow Management for Grid Computing. *Computer Journal*, 2005. Special issue of Proc. of the Workshop on Grid Performability Modelling and Measurement at the National eScience Centre, Edinburgh. N. Thomas (ed.). To appear.

[11] M. Calder, S. Gilmore, and J. Hillston. Automatically deriving ODEs from process algebra models of signalling pathways. submitted for publication, 2005.

[12] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 353–368, Vienna, May 1994. Springer-Verlag.

[13] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *TACAS'02, Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, volume 2280 of *LNCS*, pages 52–66, Grenoble, April 2002. Springer-Verlag.

[14] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In *Proc. of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351. IEEE Computer Society Press, October 2003.

[15] L.F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with Matlab*. Cambridge University Press, 2003.

[16] D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical species. *J. Comp. Phys.*, 22:403–434, 1976.

[17] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[18] M.A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Comp. Phys.*, 104:1876–1889, 2000.

[19] C. van Gend and U. Kummer. Stode: automatic stochastic simulation of systems described by differential equations. In *Proc. of 2nd International Conference on Systems Biology (ICSB2001)*, pages 326–332, California Institute of Technology, 2001.

[20] G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In *Proc. of 1st joint PAPM-PROBMIV Workshop*, volume 2165 of *LNCS*, pages 200–215. Springer-Verlag, September 2001.

[21] N. Thomas and J. Bradley. Terminating processes in PEPA. In *Proceedings of the Seventeenth UK Performance Engineering Workshop*, pages 143–154, University of Leeds, July 2001.

[22] D.T. Gillespie and L.R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *J. Comp. Phys.*, 119:8229–8234, 2003.

[23] N. Thomas and J. Hillston. Using Markovian process algebra to specify interactions in queueing systems. Technical Report ECS-LFCS-97-373, Laboratory for Foundations of Computer Science, The University of Edinburgh, 1997.

[24] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press & Pitman, ISBN 0-262-53086-4, 1989. Web page available at `http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.ps.gz`.

[25] B. Sericola. A finite buffer fluid queue driven by a Markovian queue. *Queueing Systems*, 38(2), 2001.

[26] V.G. Kulkarni and K.S. Trivedi. Fspns: Fluid stochastic petri nets. In *Proc. 14th Int. Conf. on Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 24–31. Springer-Verlag, 1993.

[27] Michael Mitzenmacher. Analyses of load stealing models based on differential equations. In *SPAA '98: Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures*, pages 212–221. ACM Press, 1998.

[28] Michael Mitzenmacher. Studying balanced allocations with differential equations. Technical Report SRC 1997–024, Digital systems research center, 1997.