GPA – a tool for fluid scalability analysis of massively parallel systems

Anton Stefanek Richard A. Hayden Jeremy T. Bradley
Department of Computing, Imperial College London
{as1005,rh,jb}@doc.ic.ac.uk

***Abstract*—Recent ordinary differential equation (ODE) based techniques allow efficient analysis of Markovian population models with extremely large state spaces. In most cases of realistic scale, they provide the only alternative to stochastic simulation. Moreover, numerical solution of the ODEs is cheaper computationally than simulation by orders of magnitude. We present the Grouped PEPA Analyser (GPA) tool with new functionality to exploit computationally inexpensive fluid analysis techniques to allow the exploration of large numbers of system configurations in models with large state spaces.**

GPA provides an efficient implementation of the fluid analysis techniques for models described in a stochastic process algebra. It implements recently developed extensions allowing specifications of complex reward measures using combinations of state based, rate accumulated and impulse rewards. Combined with the ability to efficiently capture various passage time metrics, GPA can be used to solve optimisation problems with a reward objective function under different service level agreement type constraints.

I. Introduction

Recently developed ordinary differential equation (ODE) based techniques address the *state space explosion* problem in analysing models of systems with extremely large state spaces. These can be found traditionally in models of chemical or biological systems but, due to increasing popularity of distributed and cloud computing environments, are also becoming common in performance analysis of computer systems. We present a tool *Grouped PEPA Analyser* (GPA) that implements existing fluid analysis techniques for models described in a process algebra.

GPA implements techniques to generate systems of ODEs that approximate means and higher moments of component counts in models described for the Grouped PEPA process algebra by Hayden and Bradley [1]. GPA also provides functionality allowing the calculation of passage time CDFs [2], and rate accumulated [3] and impulse rewards [4].

An older version of GPA than that described here implemented just the original ODE technique and was also used to assess the accuracy of the approximation [5]. The *PEPA Plug-in project* [6] provides fluid techniques for calculating mean populations in models described in the PEPA process algebra and the Bio PEPA framework [7] additionally provides further extensions suited to models of biological systems. In the field of chemistry, several tools exist implementing similar techniques also referred to as *moment closures* [8]. Closely related is also the framework and tool for automating *mean-field* techniques for a class of discrete time Markov chains [9].

II. GPA

GPA is a command line tool and can be obtained from http://www.doc.ic.ac.uk/~as1005/gpa. Figure 1 shows an overview of the structure of input files and the underlying software architecture. GPA input files start with definitions of

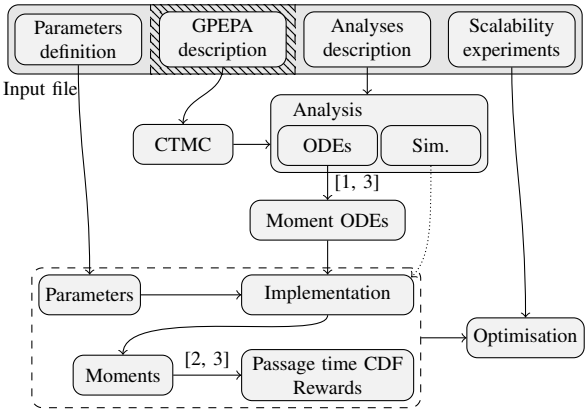

Fig. 1. Overview of GPA. Only the highlighted part depends on GPEPA.

all the constants used in the model and analyses, such as rates, initial component populations and constants used in reward functions. The following section contains a GPEPA model definition with syntax almost identical to that of GPEPA [1]. For example, a simple client/server model [4, 3] can be defined as:

```
Client          = (req,rreq).Client_waiting;
 Client_waiting = (data,rdata).Client_think;
 Client_think   = (think,rthink).Client;
Server          = (req,rreq).Server_get
                    +(break,rbreak).Server_broken;
 Server_get     = (data,rdata).Server
 Server_broken  = (reset,rreset).Server;
Clients{Client[N_c]}<req,data>Servers{Server[N_s]}
```

1) Analyses: The defined GPEPA models are then analysed. For example, an ODE analysis can be run with the command:

```
ODEs(stopTime=5.0, stepSize=0.01, density=10)
   {E[Clients:Client],Var[Servers:Server];}
```

This specifies the time horizon in which the generated ODEs are numerically solved, the time interval between successive data points and an implicit time interval given to the built-in fourth order Runge–Kutta solver. Similarly the `Simulation` analysis, where `density` is replaced by the number of independent replications, performs the stochastic simulation of the model's underlying CTMC. Each analysis can be given a list of moment based expressions to output, which are plot directly or saved as a data file with an accompanying GNUplot file.

2) Passage times: GPA allows parallel composition inside components tracked by ODEs and pattern matching on this component structure. This is useful for concise expression of certain passage time CDFs. For example, a client can be "tagged" to remember whether data has been received before, being composed with the process:

```
NotDone = (think,rt).Done    Done = (think,rt).Done
```

Inside the model definition, `Client[N_c]` is replaced by `Client<think>NotDone[N_c]`. The CDF of time until finishing the first `think` action is a sum of all components with the "absorbing" tag value `Done` [2]. For example, `%C:_<think>Done` stands for the sum of counts of the three components with the tag in the `Done` state. This CDF can be useful in expressing service level agreements (SLAs), such as requiring that the probability to finish within 15 time units is at least 0.95. Figure 2 shows a plot of such a CDF, where the SLA is satisfied.

3) Rewards: GPA supports two types of accumulated rewards. Rate rewards, useful for modelling quantities accumulated continuously over time, such as energy consumption, can be expressed as linear combinations of integrals of component counts [3], for example $\int_0^t S(u)\mathrm{d}u$ with GPA syntax `acc(S:Server)`. Impulse rewards, modelling quantities that are incremented with specific transitions of the CTMC, such as the income from finishing a transaction, can be expressed as linear combinations of action counting processes [4], for example the number of `think` actions by the time t, written as `#think`. GPA can compute moments of linear combinations of these rewards and supports a convenient variable syntax:

```
$energy = e_s*acc(S:Server)+e_sg*acc(S:Server_get)
$cost   = c_s*N_s + c_break*#break
$reward  = i_think*#think -$energy-$cost
```

See Figure 2 for a plot of the mean and standard deviation of the reward above.

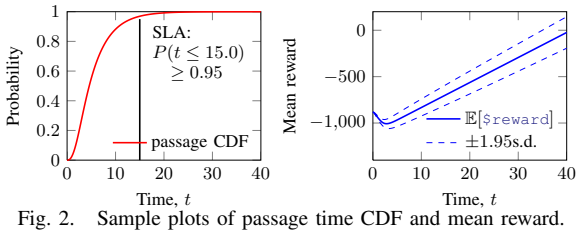

Fig. 2. Sample plots of passage time CDF and mean reward.

4) Scalability experiments: In the case of ODE analysis, GPA first generates an abstract representation of the system of ODEs and then dynamically compiles an efficient implementation. If the model structure remains unchanged, the same implementation can be used to analyse systems with different initial conditions or rate parameters. GPA uses this to provide a simple way to explore large parameter spaces. For example, a modeller can be interested in the effect of different numbers of servers and server break rate on the system behaviour. The `Iterate` command runs a given analysis for a specified discretised parameter space. For each parameter combination, it plots a moment based expression, conditioned on a constraint. Figure 3 shows an example where the expression is the mean

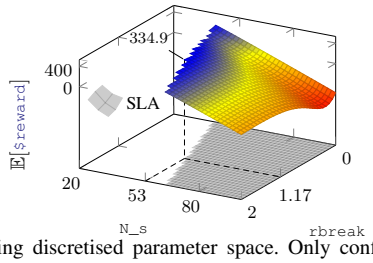

Fig. 3. Exploring discretised parameter space. Only configurations where the SLA is satisfied are considered.

of the reward above, conditioned on satisfaction of the SLA shown in Figure 2:

```
Iterate N_s from 1 to 100 with 20 steps
        rbreak from 0.0 to 2.0 with 20 steps
 ODEs(stopTime=40.0,stepSize=0.1,density=10) {
E[$reward] at 40.0 when
      E[%C:_<think>Done]/N_c at 15.0 >= 0.95}
```

GPA can output to MATLAB so that more advanced global optimisation algorithms can be used, such as the *Global search*.

III. Conclusion & Future work

GPA implements efficient fluid analysis techniques for models described in the GPEPA process algebra. The architecture of GPA is modular and offers possibilities for various further extensions. In the case of scalability experiments, more sophisticated optimisation algorithms exploiting the structure of the generated ODEs could replace the generic parameter sweeping and global optimisation algorithms.

Furthermore, GPA uses a CTMC where states are vectors of populations as an intermediate representation before generating moment ODEs. This reduces the dependence on the GPEPA process algebra to a minimum and enables application of the moment ODE techniques, and passage time and reward extensions to other formalisms that can be mapped to such CTMCs.

References

[1] R. A. Hayden and J. T. Bradley, "A fluid analysis framework for a Markovian process algebra," *Theoretical Computer Science*, vol. 411, pp. 2260–2297, May 2010.

[2] R. A. Hayden, A. Stefanek, and J. T. Bradley, "Fluid passage-time calculation in large Markov models," *in submission*, 2011.

[3] A. Stefanek, R. A. Hayden, and J. T. Bradley, "Fluid Analysis of Energy Consumption using Rewards in Massively Parallel Markov Models," in *ICPE'11*, Mar. 2011.

[4] R. A. Hayden, *Scalable performance analysis of massively parallel stochastic systems*. PhD thesis, 2011.

[5] A. Stefanek, R. A. Hayden, and J. T. Bradley, "A new tool for the performance analysis of massively parallel computer systems," in *QAPL'10*, Mar. 2010.

[6] M. Tribastone, "The PEPA Plug-in Project," in *QEST'07*, pp. 53–54, IEEE Computer Society, Sept. 2007.

[7] F. Ciocchetta and J. Hillston, "Bio-PEPA: A framework for the modelling and analysis of biological systems," *Theoretical Computer Science*, vol. 410, pp. 3065–3084, Aug. 2009.

[8] C. S. Gillespie, "Moment-closure approximations for mass-action models," *IET Systems Biology*, vol. 3, no. 1, pp. 52–58, 2009.

[9] R. Bakhshi, J. Endrullis, S. Endrullis, W. Fokkink, and B. Haverkort, "Automating the Mean-Field Method for Large Dynamic Gossip Networks," *QEST'10*, vol. 0, pp. 241–250, 2010.