Evaluating Quality of Service for Service Level Agreements

Allan Clark and Stephen Gilmore

Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh, Scotland

Abstract. Quantitative analysis of quality-of-service metrics is an important tool in early evaluation of service provision. This analysis depends on being able to estimate the average duration of critical activities used by the service but at the earliest stages of service planning it may be impossible to obtain accurate estimates of the expected duration of these activities. We analyse the time-dependent behaviour of an automotive rescue service in the context of uncertainty about durations. We deploy a distributed computing platform to allow the efficient derivation of quantitative analysis results across the range of possible values for assignments of durations to the symbolic rates of our high-level formal model of the service expressed in a stochastic process algebra.

1 Introduction

Service-oriented computing is an important focus area for industrial computer systems, highlighting the crucial interplay between service provider and service consumer. Service-level agreements (SLAs) and service policies are key issues in this domain. An SLA typically incorporates a time bound and a probability bound on a particular path through the system. It will make clear the metric against which the service is being judged, how the service provision will be measured, and the penalty to be exacted if the service is not delivered with the agreed level of quality of service (QoS). We are concerned here with the quantitative core of an SLA and wish to answer formally questions of the form "Will at least 90% of all requests receive a response within 3 seconds?" which has as a probability bound "at least 90%", as a time bound "within 3 seconds", and as the path through the system "from request to response".

An SLA needs to be established in the early specification phase for a commissioned service, and the service provider needs to ensure not later than that point in time that the SLA is credible. High-level formal modelling is helpful here because it allows us to pose precise questions about a formal model of the service to be provided and to answer them using efficient, proven analysis tools [1]. The difficulty at the early specification phase is to know whether we can match the quantitative constraints of customers' requests against the efficiency or performance of the implementation of our service. In the early specification phase in model-driven software development we have no measurement data which we can use to parameterise our high-level quantitative model (since the implementation

L. Brim et al. (Eds.): FMICS and PDMC 2006, LNCS 4346, pp. 181–194, 2007.

has not yet been built), leading to uncertainty about the values of the rate constants to be used in the computation of the passage-time quantiles needed to answer the questions about satisfaction of QoS constraints.

This uncertainty is manageable in practice because although we may not know precisely the value of the rate constants to be used in the model we may know a range of values within which they will lie. The problem then is simply to evaluate our model against our SLA measure a (possibly large) number of times. This can be done by performing a parameter sweep across the range of possible values for the rates. If each of these computations leads to the conclusion that the SLA can be met, then we can accept it even in the presence of uncertainty about the rate values. However, if any of the computations leads to the conclusion that the SLA cannot be met, then we must revise the SLA to loosen the time or probability bounds which it mandates and see if this weaker SLA is still acceptable to the service consumer. An alternative would be to try to improve some of the rates at which key activities are performed, in order to fulfil the stricter SLA and avoid the need to weaken the time or probability bounds. To help with identifying the key rates in the model we need to investigate the sensitivity of the model to changes in individual rates. To do this we evaluate our chosen measure for each rate repeatedly while varying the rate throughout its range of allowable values. This will allow us to identify those rates which have a major impact on performance if varied and those rates which impact on performance little.

Specifically, we are addressing in the present paper analysis methods and tools for the efficient computation of cumulative distribution functions (CDFs) which decide whether an SLA will be met. Set against this means of evaluating SLAs by parameter sweep is the cost of the many numerical computations needed to calculate the many CDFs required. The approach which we follow here is to evaluate simultaneously many runs of the Markov chain analyser used. Parameter sweep is an approach which falls into the class of problems commonly known as "embarrassingly parallelizable". That is, there are many independent copies of the code being run in isolation with none of the complexities of management of synchronisation points which are usually associated with parallel codes. In this setting a simple approach based on a network of workstations architecture will be effective in delivering the computational effort needed.

We used the Condor [2] high-throughput computing platform to distribute the necessary SLA computation across many hosts. Condor is a widely-used long-standing workload management system. A recent paper presenting the key ideas is [3].

We model our service in the PEPA process algebra [4]. Our models are compiled into stochastic Petri nets by the Imperial PEPA Compiler, ipc, and these are analysed by the Hydra release of the DNAmaca Markov chain analyser [5], a state-of-the-art stochastic Petri net tool which computes the passage-time quantiles needed in the computation of a CDF used in the evaluation of an SLA.

PEPA models submitted to ipc must be Cyclic PEPA [6], formed by the composition of co-operating sequential components. Each of the sequential components at the leaves of the process tree is viewed as a finite state automaton

with timed Markovian transitions and converted into a Petri net state machine. ipc then recurses back up the process tree composing these nets until it has produced a single net representing the complete PEPA model.

2 Related Work

Our use of Hydra on a distributed workload management system such as Condor is different in nature from previous work on using Hydra on distributed-memory parallel machines (examples include [7]) and distributed compute clusters (examples include [8]). One difference is that we initiate our Hydra execution from a PEPA model, via ipc, and are therefore using Markovian modelling exclusively ([8] addresses semi-Markov models). In work such as [7], [8] and [9] the emphasis is on *grande* modelling, where detailed models of systems are evaluated in the setting of many component replications. Due to the multitude of possible interleavings of the local states of each of these subcomponents it is not uncommon for such grande modelling to give rise to statespaces of order 10^6 [8], 10^7 [7], 10^8 [9], or 10^9 [10]. Although such sizes might seem modest if compared to the sizes of models analysed by *non-quantitative* procedures these dimensions place these analysis problems on the edge of tractability for Markovian analysis.

In contrast to the above, the style of modelling which we are using here is diminutive. Most nodes in our Condor cluster are typical desktop Pentium 4 PCs, with 1 CPU and with 1Gb of RAM. Each of these must be able to solve our modelling problem independently. The difference is that the prior work cited above is solving very large models a relatively small number of times whereas we are solving relatively small models a very large number of times.

An alternative method of answering the same question about SLAs would be first to encode the statement of the QoS measure as a formula in Continuous Stochastic Logic (CSL) [11] and then to model-check the formula against the PEPA model using the PRISM probabilistic symbolic model checker [12]. Computationally, this solution procedure would be very similar to the method which we employ, using uniformisation [13,14] to compute the transient analysis result needed from the continuous-time Markov chain representation underlying the PEPA model.

While this approach would have been successful for solving one run of the numerical computing procedure required we believe that we would have found difficulty in hosting multiple runs of PRISM on the Condor platform. As a batch processing system Condor has a notion of execution context called a *universe*. The ipc and Hydra modelling tools which we used run as native executables in Condor's `vanilla` universe. Java applications run on Condor's `java` universe (developed in [15]). However, PRISM combines both Java code and native C code in its use of the CUDD binary decision diagram library [16] via the Java Native Interface. The general approach to running Java code with JNI calls under Condor would be to execute the JVM under the `vanilla` universe because the `java` universe cannot guarantee to provide necessary libraries for the native code part of PRISM. However, this would in general require first copying the JVM binary onto the remote machine before execution of PRISM could begin. This

would impose a heavy penalty on run-time which would offset significantly the advantages to be gained from Condor-based distribution.

3 Markovian Process Algebras

Markovian process algebras such as PEPA extend classical process algebras by associating an exponentially-distributed random variable with each activity representing the average rate at which this activity can be performed. The random variable X is said to have an exponential distribution with parameter λ ($\lambda > 0$) if it has the distribution function

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

The mean, or expected value, of this exponential distribution is

$$\mu = E[X] = \int_{-\infty}^{\infty} x\lambda e^{-\lambda x} dx = \frac{1}{\lambda}$$

An activity in a PEPA model takes the form $(\alpha, \lambda).P$ ("perform activity α at exponentially-distributed rate λ and behave as process P"). The high-level expression of the model includes a symbolic rate variable λ. The model is evaluated against a valuation which assigns numerical values to all of the symbolic rates of the model.

All activities in a PEPA model are timed, and via the structured operational semantics of the language, PEPA models give rise to continuous-time, finite-state stochastic processes called Continuous-Time Markov Chains (CTMCs).

The relationship between the process algebra model and the CTMC representation is the following. The process terms (P_i) reachable from the initial state of the PEPA model by applying the operational semantics of the language form the states of the CTMC (X_i). For every set of labelled transitions between states P_i and P_j of the model $\{(\alpha_1, r_1), \ldots, (\alpha_n, r_n)\}$ add a transition with rate r between X_i and X_j where r is the sum of $r_1, \ldots, r_n$. The activity labels (α_i) are necessary at the process algebra level in order to enforce synchronisation points, but are no longer needed at the Markov chain level.

A CTMC can be represented by a set of states X and a transition rate matrix R. The matrix entry in position r_{ij} is λ if it is possible for the CTMC to transition from state i to state j at rate λ. An infinitesimal generator matrix Q is formed from the transition rate matrix by normalising the diagonal elements to ensure that each row sums to zero. The generator matrix is usually sparse.

3.1 Transient Analysis and Uniformisation

Investigation of SLAs requires the transient analysis of a CTMC. That is, we are concerned with finding the transient state probability row vector $\pi(t) = [\pi_0(t), \ldots, \pi_{n-1}(t)]$ where $\pi_i(t)$ denotes the probability that the CTMC is in

state i at time t. Transient and passage-time analysis of CTMCs proceeds by uniformisation [13,14]. The generator matrix, Q, is "uniformized" with:

$$P = Q/q + I$$

where $q > \max_i |Q_{ii}|$. This process transforms a CTMC into one in which all states have the same mean holding time $1/q$.

Passage-time computation is concerned with knowing the probability of reaching a designated target state from a designated source state. It rests on two key sub-computations. First, the time to complete n hops ($n = 1, 2, 3, \ldots$), which is an Erlang distribution with parameters n and q. Second, the probability that the transition between source and target states occurs in exactly n hops.

3.2 Model Checking

A widely-used logic for model checking properties against continuous-time Markov chains is Continuous Stochastic Logic (CSL) [11]. The well-formed formulae of CSL are made up of *state formulae* ϕ and *path formulae* ψ. The syntax of CSL is below.

$$\phi ::= \mathit{true} \mid \mathit{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi]$$
$$\psi ::= \mathrm{X}\,\phi \mid \phi\ \mathrm{U}^{I}\ \phi \mid \phi\ \mathrm{U}\ \phi$$

where a is an atomic proposition, $\bowtie \in \{ <, \leq, >, \geq \}$ is a relational parameter, $p \in [0, 1]$ is a probability, and I is an interval of $\mathbb{R}$. Derived logical operators such as implication ($\Rightarrow$) can be encoded in the usual way.

Paths of interest through the states of the model are characterised by the *path formulae* specified by $\mathcal{P}$. Path formulae either refer to the next state (using the X operator), or record that one proposition is always satisfied until another is achieved (the until-formulae use the U-operator).

Performance information is encoded into the CSL formulae via the time-bounded until operator (U^{I}) and the steady-state operator, $\mathcal{S}$. The evaluation of time-bounded until formulae against a CTMC in a CSL-based model checker such as PRISM [12] or MRMC [17] proceeds by transient analysis using uniformisation and a numerical procedure such as the Fox-Glynn algorithm [18].

3.3 Sensitivity Analysis

Due to the roles which activities play in creating the dynamics of our stochastic process algebra model it may be that increasing the rate of one activity increases the score obtained by the model on our chosen performance measure of interest. Conversely, increasing the rate of another activity may decrease the score which we get. Changing one rate a little may vary the score a lot. Changing another rate a lot might only vary the score a little. The study of how changes in performance depend on changes in parameter values in this way is known as *sensitivity analysis*.

Our main aim here is to determine that our SLA is met across all of the possible combinations of average values of rates across all their allowable ranges.

However, by collecting the results where one rate is varied we can examine the sensitivity of our measure with respect to that rate, at no added computational cost.

The practical relevance of sensitivity analysis is that we may find that the model is relatively insensitive to changes in one of the rates. In this case we need not spend as much effort in trying to determine precisely the exact average value of this rate. This effort would be better directed to determining the values of rates for which the model has been shown to be sensitive. Further, sensitivity analysis will identify the most critical areas to improve if failing to meet an SLA.

4 Case Study: Automotive Crash Scenario

Our case study concerns the assessment of an SLA offered by an automotive collision support service. The scenario with which these systems are concerned is road traffic accidents and dispatch of medical assistance to crash victims. Drivers wishing to use the service must have in-car GPS location tracking devices with communication capabilities and have pre-registered their mobile phone information with the service.

The scenario under study considers the following sequence of events.

- A road traffic accident occurs. The car airbag deploys.
- Deployment of the air bag causes the on-board safety system to report the car's current location (obtained by GPS) to a pre-established accident report endpoint.
- The service at the reporting endpoint attempts to call the registered driver's mobile phone.
- If there is no answer to the call then medical assistance is dispatched to the reported location of the car (presuming that the driver has been incapacitated by injuries sustained in the accident).

There may be many possible reasons why the driver does not answer the phone. The phone may be turned off; its battery may be flat; the phone may be out of network range; the driver may have switched to a new telephone provider, and not informed the collision support service; the phone may not be in the car; it may have been smashed on impact; or many other possibilities.

The accident reporting service cannot know the exact reason why the driver does not answer the phone. They only know that an accident has happened which was serious enough to cause the airbag to be deployed, and that the driver has not confirmed that they do not need medical assistance. In this setting they will dispatch medical help (even if sometimes this will mean that help is sent when it is not absolutely necessary).

The SLA related to this scenario concerns the response time of the passage from the deployment of the airbag to the dispatch of medical assistance. The parameters of our modelling study are:

- the rate at which information on the location of the car—and any other pertinent information such as speed on impact, engine status, and other

diagnostic information obtained from the on-board diagnostic systems and controllers—can be reported to the accident reporting service;
- the time taken to confirm that the driver is not answering their mobile telephone; and
- the time taken to contact the emergency services to dispatch medical assistance.

None of these parameters are known exactly, but their average values are known to lie within a range of acceptable operation. We are, of course, interested in worst case bounds on passage-time quantiles and also in best case analysis but also in the variety of possible responses in between.

4.1 PEPA Model

In this section we consider the sequence of events which begins with the deployment of the airbag after the crash and finishes with the dispatch of the medical response team. The first phase of the sequence is concerned with relaying the information to the remote service, reporting the accident. When the diagnostic report from the car is received the service processes the report and matches it to the driver information stored on their database.

$$Car_1 \stackrel{def}{=} (airbag, r_1).Car_2$$
$$Car_2 \stackrel{def}{=} (reportToService, r_2).Car_3$$
$$Car_3 \stackrel{def}{=} (processReport, r_3).Car_4$$

The second phase of this passage through the system focuses on the attempted dialogue between the service and the registered driver of the car. We consider the case where the driver does not answer the incoming call because this is the case which leads to the medical response team being sent.

$$Car_4 \stackrel{def}{=} (callDriversPhone, r_4).Car_5$$
$$Car_5 \stackrel{def}{=} (timeoutDriversPhone, r_5).Car_6$$

The service makes a final check on the execution of the procedure before the decision is taken to send medical help. At this stage the driver is awaiting rescue.

$$Car_6 \stackrel{def}{=} (rescue, r_6).Car_7$$
$$Car_7 \stackrel{def}{=} (awaitRescue, r_7).Car_1$$

This takes us to the end of the passage of interest through the system behaviour.

4.2 Rates Constants and Ranges

All timings are expressed in minutes, because that is an appropriate granularity for the events which are being modelled. Thus a rate of 1.0 means that something happens once a minute (on average). A rate of 6.0 means that the associated activity happens six times a minute on average, or that its mean or expected duration is ten seconds, which is an equivalent statement. A table of the ranges of average rate values used appears in Table 1.

4.3 Sensitivity Analysis for the Automotive Crash Scenario

We consider how the cumulative distribution function for the passage from airbag deployment to dispatch of medical assistance is affected as the values of the rates r_2 to r_6 are varied as specified in Table 1. The results are presented in Figure 1.

Table 1. Minimum and maximum values of the rates from the model

Rate	Value min	Value max	Meaning
r_1	600.0	600.0	an airbag deploys in 1/10 of a second
r_2	2.0	10.0	the car can transmit location data in 6 to 30 seconds
r_3	0.5	1.5	it takes about one minute to register the incoming data
r_4	1.5	2.5	it takes about thirty seconds to call the driver's phone
r_5	1.0	60.0	give the driver from a second to one minute to answer
r_6	0.25	3.0	vary about one minute to decide to dispatch medical help
r_7	1.0	1.0	arbitrary value — the driver is now awaiting rescue

What we see from these results is that variations in upstream rates (near the start of the passage of interest) such as r_2, r_3 and r_4 have less impact overall than variations in downstream rates (near the end of the passage of interest) such as r_5 and r_6. This is true even when the scale over which the upstream rates are varied is much more than the scale over which the downstream rates are varied (for example, contrast variation in r_2 against variation in r_6).

The conclusion to be drawn from such an observation is that, if failing to meet a desired QoS specified in an SLA then it is better to expend effort in making a faster decision to dispatch medical help (governed by rate r_6) than to expend effort in trying to transmit location data faster (governed by rate r_2), over the range of variability in the rates considered in the present study.

Another use of this sensitivity data would be to find an optimum time to hold while waiting for the driver to answer the phone. The optimisation problem to be solved here is to decide how long to wait before terminating the call in case of non-answer. If the service providers wait too long then they risk failing to meet their SLA. If they wait too little then they risk dispatching medical assistance when it is not actually necessary. In this case the sensitivity graph of rate r_5 shows a portion where changes in rate value have little impact and so targeting the lowest rate here gives the driver more time to answer the phone.

A further kind of graph which can be drawn is depicted in Figure 2. To produce this graph we have held constant the time and varied two of the rates involved, r_5 and r_6. From this kind of graph one can analyse how the probability of completion by a chosen time bound can depend on the relationship between two of the rates. In this graph we can see that when the rate r_5 is low, as in the front line of the graph, then varying the rate r_6 has little effect. However the back line of the graph shows that when rate r_5 is high, varying rate r_6 has a greater effect.

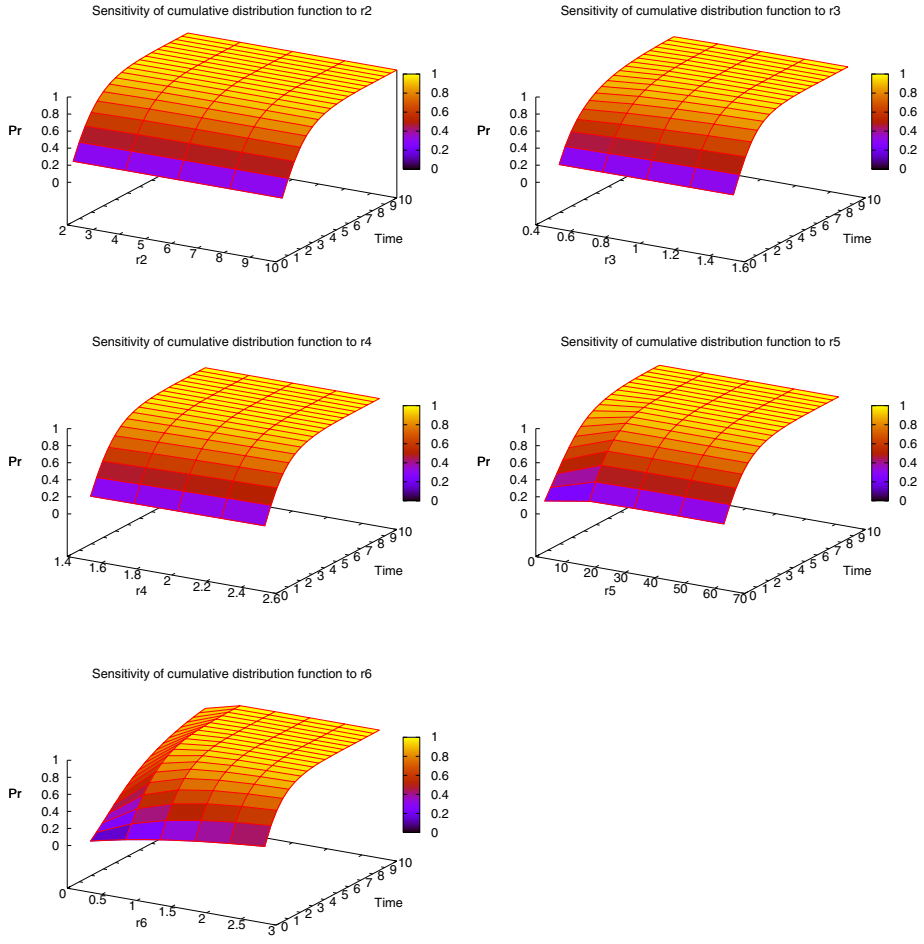

Fig. 1. Graphs of cumulative distribution function sensitivity to changes in rates for the passage from airbag deployment to dispatch of medical assistance

The reverse relationship between rates r_5 and r_6 is also true. The model we used was a linear model, which means that there were few paths through the model. In particular the action *rescue* governed by the rate r_6 cannot be performed until the action *timeoutDriversPhone*, regulated by rate r_5, has occurred. Also once the *timeoutDriversPhone* action has occurred there is nowhere for the model to go but to a *rescue* action. This means that if either of the two rates associated with these two actions is very low, then that action will be the bottleneck for that part of the model. Varying the other rate will have less effect.

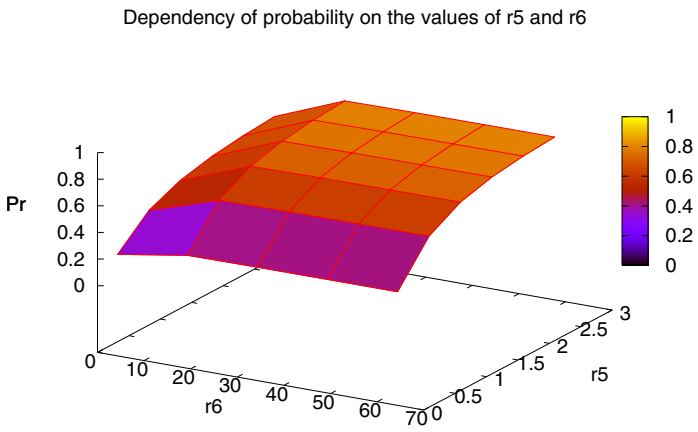

Fig. 2. Graph of probability of completion against variation in the rates r_5 and r_6, for a fixed time value

5 Relation to Model Checking

In this section we consider how the results expressed above relate to model checking a CSL formula against our model of the system. Expressed as a CSL formula an example of the kind of question which we are asking is the following.

$$airbag \Rightarrow \mathcal{P}_{>0.9}[true\ \mathrm{U}^{[0,10]}\ rescue]$$

In words, this says "If the airbag in the car deploys, is it true with probability at least 0.9 that the rescue service will be sent within 10 minutes?"

We consider a more general form of the question which is the following

$$airbag \Rightarrow \mathcal{P}_{\bowtie p}[true\ \mathrm{U}^{[0,10]}\ rescue]$$

We consider this for all relations $\bowtie \in \{<, \leq, >, \geq\}$ and for all values of the probability bound $0 \leq p \leq 1$. Further, we answer these general formulae not for only a single assignment of values to symbolic rate variables (as would be the case for conventional model checking) but across the range of assignments presented in Figure 1.

In order to determine upper and lower bounds on the probability with which the rescue service is dispatched within 10 minutes we can simply plot the probability computed via transient analysis against experiment number. Each mapping of rate values onto symbolic rate names is an experiment.

The graph of computed probability against experiment number for the first fifty experiments is shown in Figure 3. Experiments are grouped whereby a group contains about five evaluations of the CDF corresponding to the SLA for five assignments of concrete rate values to one of the symbolic rates r_2 to r_6. This shows slightly more than the first eight groups of experiments.

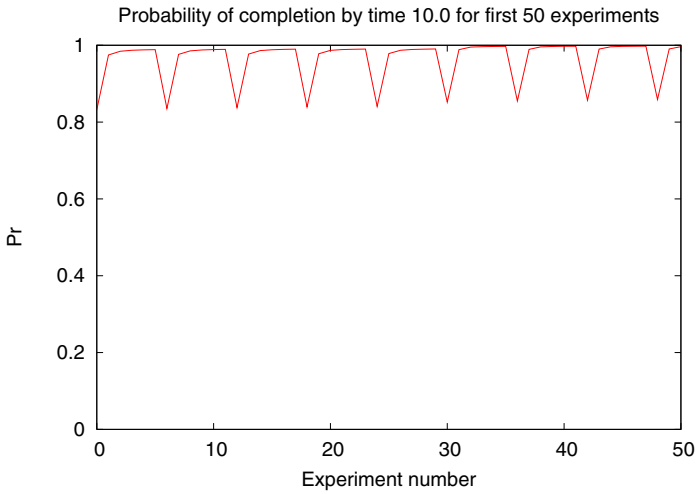

Fig. 3. Graph of probability of completing the passage from airbag deployment to medical assistance dispatch within ten minutes plotted against experiment number over the first fifty experiments

The graph of computed probability against experiment number for all the 3750 experiments is shown in Figure 4. At this level of granularity it is not easy to pick out groups of runs but one can see that all experiments achieve at least a minimum QoS that at least 83% of calls to the service will lead to medical assistance being dispatched within 10 minutes.

One use of these graphs is to identify all of the combinations of average rate values which allow the service to satisfy an SLA which requires their quality of service to be above a specific threshold. For example, say that the service providers wish to, or need to, meet the SLA that the rescue service is dispatched within 10 minutes in 92% of cases of airbag deployment. The graph in Figure 4 identifies all of the combinations of parameter values which achieve this bound, or do better. Some of these might be much easier to realise than others so the service could meet its QoS requirement by striving for those combinations of average rates for individual actions of the system such as taking the decision to dispatch medical help (at rate r_6).

6 Further Work

Our future programme of work on using ipc and Hydra on the Condor distributed computing platform is directed towards making better use of the support which Condor provides for distributed computing. This will include the use of the `standard` universe which will allow checkpointing within a run, and allow a long-running Hydra computation to be migrated in-run from a machine claimed by a user onto a presently-idle machine.

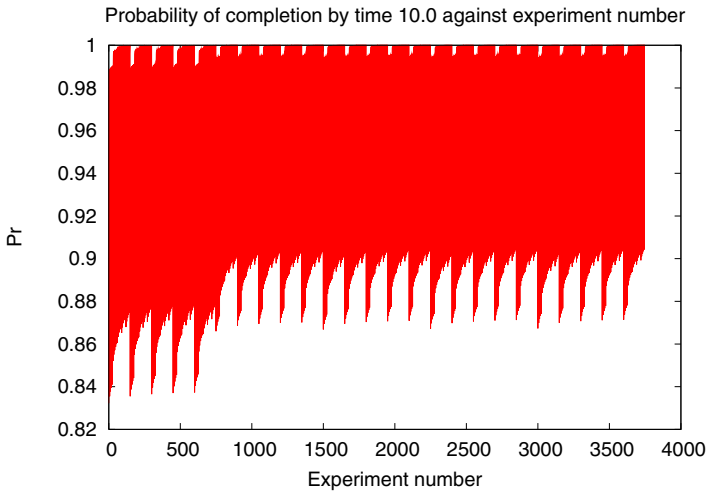

Fig. 4. Graph of probability of completing the passage from airbag deployment to medical assistance dispatch within ten minutes plotted against experiment number over all 3750 experiments

In this work we have made the conceptually convenient simplification of thinking of Hydra as a single, indivisible application which accepts a stochastic Petri net as input and returns as its output a CDF showing passage-time quantiles. While this is an accurate conceptual description Hydra is in fact structured as a collection of independent components (a parser, a state-space generator, a functional analyser, a solver and a uniformiser). The application which we think of as Hydra is a high-level driver executing these components in the order described above.

The opportunity which this gives us for the future is to structure Hydra as a directed acyclic graph (DAG) of component tasks. To run Hydra on Condor in this way we would specify the inputs and outputs from each sub-component (state-space generator, functional analyser and others) and connect these together replacing Hydra's top-level driver with the appropriate use of Condor's DAG manager (DAGman). This would offer a greater range of possibilities for component deployment on our Condor pool.

7 Conclusions

The automotive rescue case study used in this paper gives rise to a relatively small continuous-time Markov chain, the unit solution cost of which is not excessive. However, when repeatedly re-running this solution procedure for different parameter values these small costs quickly start to add up. The Condor distributed computing system allowed us to execute these many copies of the job simultaneously.

The parallel structure of the joint computation was very simple; running a sequential application multiple times. No dynamic process creation was required within an individual run, and no inter-process communication was needed. A full-blown parallel computing infrastructure such as PVM or MPI would have been excessive but Condor suited our problem very well.

The style of analysis which we pursue here is embarrassingly parallelizable, meaning that the throughput of jobs increases linearly with the number of machines available. This means that if given access to a larger Condor pool, or the ability to connect Condor pools together, then the rate at which jobs can be processed continues to grow and is not capped by an inherent bound on problem scalability. Thus the combination of ipc, Hydra and Condor as a modelling and experimentation framework provides a strong platform on which to conduct larger and more complex experiments.

Acknowledgements

The authors are supported by the SENSORIA project (EU FET-IST Global Computing 2 project 016004). We are grateful to Angelika Zobel and Nora Koch of F.A.S.T. München for the specification of the automotive case study. We modified the open-source software tool ipc developed and made freely available by Jeremy Bradley. We ran our models on the Condor cluster provided in the School of Informatics at Edinburgh and benefited from advice from Chris Cooke on using this effectively.

References

1. William J Knottenbelt. Generalised Markovian analysis of timed transition systems. MSc thesis, University of Cape Town, South Africa, July 1996.
2. Condor project homepage. Website with documentation and software, University of Wisconsin-Madison, April 2006. `http://www.cs.wisc.edu/condor/`.
3. Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
4. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
5. J.T. Bradley and W.J. Knottenbelt. The ipc/HYDRA tool chain for the analysis of PEPA models. In *Proc. 1st International Conference on the Quantitative Evaluation of Systems (QEST 2004)*, pages 334–335, Enschede, Netherlands, September 2004.
6. J. Hillston and M. Ribaudo. Stochastic process algebras: a new approach to performance modeling. In K. Bagchi and G. Zobrist, editors, *Modeling and Simulation of Advanced Computer Systems*. Gordon Breach, 1998.
7. Nicholas J Dingle, Peter G Harrison, and William J Knottenbelt. Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models. *Journal of Parallel and Distributed Computing*, 64:908–920, 2004.

8. Jeremy T Bradley, Nicholas J Dingle, Peter G Harrison, and William J Knottenbelt. Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models. In *Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, Nice, April 2003. IEEE Computer Society Press.
9. W J Knottenbelt, P G Harrison, M S Mestern, and P S Kritzinger. A probabilistic dynamic technique for the distributed generation of very large state spaces. *Performance Evaluation*, 39(1–4):127–148, February 2000.
10. R. Mehmood and Jon Crowcroft. Parallel iterative solution method for large sparse linear equation systems. Technical Report UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, UK, October 2005.
11. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, volume 1102 of *LNCS*, pages 169–276. Springer-Verlag, 1996.
12. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In A.J. Field and P.G. Harrison, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 200–204, London, UK, April 2002. Springer-Verlag.
13. W. Grassmann. Transient solutions in Markovian queueing systems. *Computers and Operations Research*, 4:47–53, 1977.
14. D. Gross and D.R. Miller. The randomization technique as a modelling tool and solution procedure for transient Markov processes. *Operations Research*, 32:343–361, 1984.
15. Al Globus, Eric Langhirt, Miron Livny, Ravishankar Ramamurthy, Marvin Solomon, and Steve Traugott. JavaGenes and Condor: Cycle-scavenging genetic algorithms. In *Proceedings of the ACM Conference on Java Grande*, pages 134–139, San Francisco, CA, 2000.
16. F. Somenzi. *CUDD: CU Decision Diagram Package*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, February 2001.
17. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Proceedings of the Second International conference Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE CS Press, 2005.
18. Bennett L. Fox and Peter W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31:440–445, 1988.